UNIVERSITY OF ALBERTA

Structure and Randomness of Satisfiability

by

Călin-Doru Anton ©

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta

Fall 2005

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

0-494-08608-4

Your file *Votre référence*
ISBN
Our file *Notre reference*
ISBN

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

This dissertation contains three research works focusing on structural properties of Satisfiability. The main goal is to better understand, characterize, classify and compare Satisfiability instances. In the first part a lower bound for the unsatisfiability of random Generalized Satisfiability generated over a set of globally consistent functions is established. The result is obtained using the method of differential equations introduced by Wormald and used by Achlioptas to prove similar lower bounds for k-SAT. The lk-SAT model is introduced and analyzed in the second part. We prove upper and lower bounds for the lk-SAT threshold. The upper bound is obtained using the second moment method, while the lower bound is an application of the result proved in first part. The lk-SAT model enforces a tun-able uniform type of clustering on the selection of clauses. The higher local dependency among variables induced by the model is used to highlight some limitations of a few state of the art solvers. In the last part, the same input length criterion for comparing Satisfiability instances is investigated. This criterion is used to derive several conclusions which contrast common perceptions based on comparing the difficulty of Satisfiability instances which have the same number of variables. It is shown that under the same input length criterion 3-SAT is the hardest of k-SAT and direct encoding produces harder instances than log encoding, when applied to random Subgraph Isomorphism Problem.

ACKNOWLEDGEMENT

To all those who helped me. May God bestow His grace upon them.

Table of Contents

1 Introduction 1

2 A lower bound for the unsatisfiability of a subclass of Generalized Satisfiability 3

2.1 Introduction . . . 3

2.2 Definitions and Notation . . . 4

2.3 Related Work . . . 9

2.3.1 Complexity . . . 9

2.3.2 Random Models and Their Properties . . . 10

2.3.3 Lower Bounds for Unsatisfiability . . . 16

2.4 A lower bound for unsatisfiability . . . 18

2.5 Proof of the result . . . 21

2.5.1 The process graph and its properties . . . 21

2.5.2 Proof of Theorem 2.4 . . . 24

2.6 Conclusions and Future Work . . . 39

3 A variation of the random k-SAT model 43

3.1 Introduction . . . 43

3.2 General framework and definitions . . . 44

3.3 Related work . . . 46

3.3.1 NK-landscape 46
3.3.2 SAT instances 47
3.3.3 SAT solvers 53
3.4 Evolution for fixed l - theoretical Results 58
3.4.1 A theoretical upper bound for satisfiability 59
3.4.2 A lower bound for unsatisfiability 63
3.5 Evolution for fixed l - Empirical results 65
3.5.1 Empirical characterization of the model 66
3.5.2 Comparison of solvers' performances 67
3.5.3 Adding the preprocessing procedure 77
3.6 Evolution with l 80
3.7 Minimum unsatisfiable sets of lk-SAT instances 83
3.8 Conclusions 85

4 On Comparing CNF SAT instances **92**
4.1 Introduction 92
4.2 Motivation 93
4.3 Definitions, background knowledge and related work 95
4.3.1 Binary Constraint Satisfaction Problem 96
4.3.2 Subgraph Isomorphism 96
4.3.3 The DCS SAT solver 103
4.4 Comparison of random k-SAT 104
4.5 Log-encoding vs direct-encoding for random SGI 111
4.6 Log encoding of random SGI vs random k-SAT 115
4.7 Conclusions 118

5 Conclusions **122**

A An approximation of the size of the instances for the trivial upper bound **124**

B Characterization of the hardest instances **127**

B.1 Direct encoding VS log encoding results 127

B.2 Log encoding VS k-SAT results 129

B.3 Index of definitions and notations 136

List of Tables

3.1 Empirical 50% satisfiability points 66

3.2 Values used in the experiments 69

3.3 Classification of complete solvers, according to solving time 69

3.4 Percentage of satisfiable instances solved by each solver, for $k = 3$, $l = 3$ 74

3.5 Percentage of satisfiable instances solved by each solver, for $k = 3$, $l = 4$ 74

3.6 Percentage of satisfiable instances solved by each solver, for $k = 3$, $l = 5$ 75

3.7 Percentage of satisfiable instances solved by each solver, for $k = 4$, $l = 4$ 76

4.1 "Contradictory" conclusions derived under different comparison criteria. 118

A.1 Size of instances for trivial unsatisfiability 126

B.1 Characterization of the hardest instances for BerkMin 128

B.2 Characterization of the hardest instances for kcnfs 130

B.3 Characterization of the hardest instances for satz 131

B.4 Characterization of the hardest instances for DCS 132

B.5 Characterization of the hardest instances for antisatz 133
B.6 Instance size and median running time (sec) for log encodings of the hardest random SGI - graph of order 16. 134
B.7 Instance size and median running time (sec) of 8-SAT at threshold. . 135

List of Figures

2.1 Phase transition of 3-SAT 12

2.2 The easy-hard-easy pattern for 3-SAT 12

2.3 The Unit Clause Algorithm 27

3.1 "Generic" DPLL procedure 55

3.2 "Generic" stochastic SAT solver 56

3.3 Evolution of satisfiability with m 68

3.4 Evolution of difficulty with m 71

3.5 Evolution of difficulty with n, above and below threshold for $k = 3, l = 3$ 72

3.6 Evolution of difficulty with n, at the 50% satisfiability 72

3.7 Scatter plot of ST for kcnfs versus ST for P&kcnfs for random 3-SAT 79

3.8 Scatter plot of ST for Berkmin versus ST for P&Berkmin for random 3-SAT 79

3.9 Evolution of hardness with l ($n = 400$) 81

3.10 Evolution of hardness with l and n ($m/n = 4.30$) 81

3.11 Different patterns in variation of hardness with l for $m/n < 4.3$ - left and $m/n > 4.3$ - right ($N = 400$) 83

3.12 Different evolution of ST and SM (left), ST and NVN (right) 84

4.1 Evolution of hardness (expressed by time) with size for complete solvers 105

4.2 Evolution of hardness (expressed by time) with size for stochastic solvers 107
4.3 Evolution of hardness (expressed by number of nodes) with size for complete solvers 108
4.4 Evolution of hardness (expressed by number of steps) with size for stochastic solvers 109
4.5 Evolution of hardness (expressed by time) with size for the hardest classes of log encoding and direct encoding of random SGI. 114
4.6 Evolution of hardness (expressed by time) with size for log encoding of hardest random SGI and 8-SAT. 117

Chapter 1

Introduction

Satisfiability is the prototypical problem for Artificial Intelligence. Despite having a high complexity, NP-complete, it has a very simple formulation: find a truth assignment which satisfies a given boolean formula.

After being proved NP-complete, Satisfiability attracted the interest of many researchers resulting in thousands of papers. Recently, an electronic journal - Journal on Satisfiability, Boolean Modeling, and Computation- has been dedicated entirely to this problem.

Beyond its appeal as a possible candidate for finding a solution to the thirty year old dilemma "P=NP", it has many practical applications. Formal verification of software and hardware reduces to checking if a boolean formula is satisfiable. Therefore, it has attracted a lot of interest from the big producers of chip-sets: IBM, Intel, AMD. The quickest method of solving some practical problems, where planning is the best example, is or used to be, to apply a Satisfiability solver to their equivalent boolean formulas. Many advanced features of the direct solvers of such problems have been inspired by the performances of the Satisfiability solvers on the corresponding boolean formula. Hence, any advance in Satisfiability solving has the potential to boost the solvers of other practical problems.

There is a permanent competition between finding new, challenging Satisfiability instances and designing efficient Satisfiability solvers able to tackle the challenges. This competition has resulted in many improvements of the Satisfiability solvers in the last decade.

We believe that producing new challenging instances is not enough for improving solvers - characterizing and classifying the new instances is also necessary. This ensures that any progress made in solving some instances can be used (or adapted) to solve an entire class of instances. The first moves in this direction are to investigate the structural properties of these instances and to assess the influence of these properties on the instances difficulty.

This thesis should be viewed as a step in this direction. It has three main parts, one analyzing uniform random instances, one investigating the influence of local dependency of variables on instance hardness, and one studying a way of comparing different instances.

The main contributions of this thesis are:

- establish a lower bound for the unsatisfiability of a subclass of random generalized Satisfiability;
- introduce and analyze a new model for generating Satisfiability instances, which highlights some limitations of some state of the art solvers;
- investigate the same input length criterion for comparing Satisfiability instances.

To each of the three parts described above a self-contained chapter is dedicated. We preferred this form of presentation for two reasons: each part presents an independent aspect of an investigation in structural properties of Satisfiability and each chapter is a stand alone piece of research in different stages of publication.

Chapter 2

A lower bound for the unsatisfiability of a subclass of Generalized Satisfiability

2.1 Introduction

The Boolean Constraint Satisfaction Problem (BCSP), also known as Generalized Satisfiability is regarded as an intermediate between the Satisfiability(SAT) and Constraint Satisfaction Problems(CSP). It resembles Satisfiability as the variables take boolean values, and Constraint Satisfaction Problem as it allows general constraints. In the last few years, Random Boolean Constraint Satisfaction has attracted the interest of many researchers [13, 12, 18, 14].

Several results obtained for SAT were extended to Boolean CSP and eventually to CSP. Similarly we expect that our result has the potential to be extended to CSP at some future point.

In this chapter we give a lower bound for the unsatisfiability of a subclass of random symmetric BCSP generated over a set of globally consistent functions. Our

result is an adaptation of the technique used by Achlioptas for obtaining lower bounds for the uniform random k-SAT threshold [2].

The same technique was used to obtain a lower bound for uniform random 3-NAE SAT [3], and to obtain the exact threshold for $\mathcal{F}_{n,M}^{2,3,2}$ [23]. To the best of our knowledge, this is the first research work that uses an algorithmic approach to prove a lower bound for a large class of CSP's. While the algorithm used in this paper is rather simplistic, the methodology may be extended to better algorithms, which may result in tighter lower bounds.

2.2 Definitions and Notation

We say that a property holds *with high probability (whp)* if asymptotically (usually for $n \to \infty$) it holds with probability 1. We say that a property holds *with positive probability (wpp)* if asymptotically it holds with probability bounded away from 0.

Given a boolean function f, we will use $ar(f)$ to denote the arity of f and $AR(f)$ to denote the set $\{1, \ldots, ar(f)\}$. A *nogood* of f is a tuple $(i_1, \ldots, i_{ar(f)}) \in \{0,1\}^{ar(f)}$ such that $f(i_1, \ldots, i_{ar(f)}) = 0$. We will use $ng(f)$ to denote the number of nogoods of f, and $\kappa(f) = \frac{ng(f)}{2^{ar(f)}}$ to denote the density of nogoods of f. A boolean function is *empty* if $ng(f) = 2^{ar(f)}$. It is *trivial* if $ng(f) = 0$. We will use $\mathbb{T}$ to denote the set of all trivial functions, regardless of their arity. A boolean function f or arity k is *satisfied* by $x_1, \ldots, x_k$ if and only if $f(x_1, \ldots, x_k) = 1$. A boolean function is *1-valid (0-valid)* if it is satisfied by setting all its variables to 1 (0), that is to say $f(1, \ldots, 1) = 1$ ($f(0, \ldots, 0) = 1$). We will use $\bigvee$ to denote the boolean functions (regardless of their arity) which are satisfied only when at least one of their arguments is 1. For example $\bigvee(x, y, \neg z)$ is 1 if one of x, y or $\neg z$ takes value 1.

A *Boolean Constraint Satisfaction Problem (BCSP)* consists of a finite set of boolean variables $\{v_1, \ldots, v_n\}$, and a finite set of boolean constraints. Each such

constraint denotes the allowed combinations of values for the variables it affects. A solution is an assignment of a value to each variable such that all constraints are satisfied. A typical BCSP asks to determine if a solution exists, to find a solution, or to find all solutions.

A boolean constraint C on the ordered set of distinct variables $(v_1^C, \ldots, v_k^C)$ is defined by a boolean function f_C:$\{0,1\}^k \rightarrow \{0,1\}$ and a tuple $(b_1^C, \ldots, b_k^C)$ of literals where for each $j \in \{1, \ldots, k\}$, b_j^C is either v_j^C or $\neg v_j^C$. We will use $f_C[b_1^C, \ldots, b_k^C]$ to denote such a constraint. An assignment Φ: $\{v_1, \ldots, v_n\} \rightarrow \{0,1\}$ satisfies such a constraint C if $f_C(\Phi(b_1^C), \ldots, \Phi(b_k^C)) = 1$, where $\Phi(b_i) = \Phi(v_i)$ if $b_i = v_i$; otherwise $\Phi(b_i) = 1 - \Phi(v_i)$. We say that x_i is a *positive variable (negative variable)* of a constraint C if $C = f[b_1, \ldots, b_k]$ and $b_i = x_i$ ($b_i = \neg x_i$), where f is some boolean function and k is its arity.

Observation 2.1 A boolean constraint can be defined by several combinations of boolean functions and literals. For example the constraint $\{(x = 0, y = 0), (x = 1, y = 1)\}$ can be expressed by $f[x, y]$ where $f^{-1}(1) = \{(0,0), (1,1)\}$ or $g[\neg x, y]$ where $g^{-1}(1) = \{(1,0), (0,1)\}$.

A *clause* of size k over the set of literals $(b_1, \ldots, b_k)$ is a boolean constraint of the type $b_1 \vee b_2 \vee \cdots \vee b_k$, where $\vee$ is the boolean or operator. Notice that the clause $b_1 \vee b_2 \vee \cdots \vee b_k$ can be denoted by $\bigvee(b_1, \ldots, b_k)$. Any boolean function may be expressed as a formula that is a conjunction of disjunctions and such a formula is said to be in *Conjunctive Normal Form (CNF)*. A CNF formula is *weakly positive (weakly negative)* if each of its clauses has at most one negative variable (positive variable).

A boolean constraint C *strongly depends on a literal* b, if and only if b has a value of 1 in every satisfying assignment of C. A boolean constraint C *strongly depends on a 2-XOR relation* B, if and only if every satisfying assignment of C satisfies B.

Let S_k denote the permutation group of k elements, id_k be the identity permutation of k elements, i.e. id_k=$(1,\ldots,k)$, and $\overrightarrow{id_2^k}$ be a vector consisting of k repetitions of id_2.

We use the framework of [13] to define equivalence classes of the constraints. Let:

- f be a boolean function, and $k = ar(f)$;
- σ be a permutation, $\sigma \in S_k$;
- $\vec{\tau}$ be a sign vector (polarization function), $\vec{\tau} \in (S_2)^k$, $\vec{\tau} = (\tau_1,\ldots,\tau_k)$ where for each $i = 1,\ldots,k$, $\tau_i : \{0,1\} \to \{0,1\}$ is a permutation of the Boolean domain.[1]

We denote by $f_\sigma^{\vec{\tau}} : \{0,1\}^k \to \{0,1\}$ the function

$$f_\sigma^{\vec{\tau}}(a_1,\ldots,a_k) = f(\tau_1(a_{\sigma(1)}),\ldots,\tau_k(a_{\sigma(k)})).$$

Let $\tilde{f}$ denote the closure of f under permutations and vector signs:

$$\tilde{f} = \{f_\sigma^{\vec{\tau}} | \sigma \in S_k \text{ and } \vec{\tau} \in (S_2)^k\}.$$

For any set F of boolean functions let $\tilde{F}$ denote $\bigcup_{f \in F} \tilde{f}$.

Observation 2.2

- If $\tilde{f} \cap \tilde{g} \neq \emptyset$ then $\tilde{f} = \tilde{g}$;
- For any $f \in \tilde{f}$ and any $f' \in \tilde{f}$, $ar(f) = ar(f')$, thus extending the notation we define $ar(\tilde{f}) = ar(f)$, for $f \in \tilde{f}$, and $AR(\tilde{f}) = AR(f)$, for $f \in \tilde{f}$;
- If $\tilde{f} \cap \mathbb{T} \neq \emptyset$, then $\tilde{f} \subset \mathbb{T}$.

[1] The notational abuse is "justified" by the isomorphism between the permutation group of the boolean domain and S_2.

A *family of distinct boolean functions* $\mathcal{F}$ is a pair (F, P), where F is a set of boolean functions s.t. $\forall\{f \neq g\} \subset F,\ \tilde{f} \neq \tilde{g}$, and P is a probability distribution over F which assigns a positive probability to each member of F.

Let $\mathcal{F}$ be a family of distinct boolean functions, n, m be positive integers. $BCSP_{n,m}(\mathcal{F})$ is a random model for boolean CSP with n variables and m constraints. Each constraint C_i is defined as follows:

- select a constraint function f from F, according to P;
- select an ordered set $(v^i_1, \ldots, v^i_{ar(f)})$ of distinct variables uniformly at random from the set of all variables;
- select a permutation σ_i uniformly at random from $S_{ar(f)}$;
- select a sign vector $\vec{\tau^i}$ uniformly at random from $(S_2)^{ar(f)}$;
- let C_i be $f^{\vec{\tau^i}}_{\sigma_i}[v^i_1, \ldots, v^i_{ar(f)}]$.

Observation 2.3 Each C_i is equivalently defined by

$$f^{\vec{\tau_i}}_{id_{ar(f)}}[v^i_{\sigma(1)}, \ldots, v^i_{\sigma(ar(f))}] \text{ or by}$$
$$f^{\overrightarrow{id_2^{ar(f)}}}_{id_{ar(f)}}[\ b^i_{\sigma(1)}, \ldots, b^i_{\sigma(ar(f))}],$$

where for each $j \in AR(f)$, $b^i_{\sigma(j)} = \begin{cases} v^i_{\sigma(j)}, & \text{if } \tau^i_j = id_2 \\ \neg v^i_{\sigma(j)}, & \text{otherwise} \end{cases}$

For any instance of $BCSP_{n,m}(\mathcal{F})$ and any $f \in F$, let

$$C_f = \{C_i | C_i = f^{\vec{\tau_i}}_{\sigma_i}[x^i_1, \ldots, x^i_{ar(f)}] \text{ and } f \in \tilde{f}\} \text{ and } c_f = |C_f|.$$

Observation 2.4 For any instance of $BCSP_{n,m}(\mathcal{F})$ and any $f \in F$, the constraints in C_f are independent and equally likely. The vector $(c_f)_{f \in F}$ is a multidimensional random variable following a multinomial distribution $\mathbf{Mn}(m, (P(f))_{f \in F})$.

Given a boolean function f, $i \in AR(f)$ and $e \in \{0,1\}$, (as in [13]) we will use $\prod_i^e(f)$ to denote the boolean function arity $ar(f)$-1, defined by:
$\prod_i^e(f)(b_1,\ldots,b_{ar(f)-1}) = f(b_1,\ldots,b_{i-1},e,b_i,\ldots,b_{ar(f)-1})$.
A *globally consistent boolean function* is a boolean function having the property that for any $j \in \{1,\ldots,ar(f)-1\}$, any $\{i_1 < i_2 < \cdots < i_j\} \subset AR(f)$ and any $\{e_1,\ldots,e_j\} \in \{0,1\}^j$, $\prod_{i_1}^{e_1}(\prod_{i_2}^{e_2}(\ldots(\prod_{i_j}^{e_j}(f))\ldots))$ is nonempty. A *globally consistent boolean constraint function* (GCBC) is a globally consistent boolean function with the property that for any $i \in AR(f)$, $\prod_i^0(f) \neq \prod_i^1(f)$ (it depends on all its variables).

Observation 2.5 If f is GCBC of arity at least 2, then it is nonempty, nontrivial, and for each $i \in AR(f)$:

- $\prod_i^1(f)$ and $\prod_i^0(f)$ are either both GCBC or one is GCBC and the other is trivial.

- $ng(f) = ng(\prod_i^1(f)) + ng(\prod_i^0(f))$.

Observation 2.6 It is easy to see that any GCBC of arity larger than 2, does not strongly depend on a literal or on a 2-XOR relation.

Observation 2.7 For any boolean function f, any $f_1 \in \tilde{f}$, any $f_2 \in \tilde{f}$, and any $(i,e) \in AR(\tilde{f}) \times \{0,1\}$, $\prod_i^e(f_1) = g$ if and only if $\exists(i',e') \in AR(\tilde{f}) \times \{0,1\}$ s.t. $\prod_{i'}^{e'}(f_2) \in \tilde{g}$.

Even though the globally consistent property is very restrictive, there are GCBC functions which have as many as $2^{ar(f)-1}$ nogoods. For example f defined by $f^{-1}(0) = \{(0,0,0),(0,1,1),(1,0,1),(1,1,0)\}$ has 4 nogoods and is GCBC.

2.3 Related Work

Almost all results regarding BCSP were obtained by extending similar results from SAT. For this reason in this section we will introduce the results from an historical point of view, presenting their evolution from SAT to BCSP.

2.3.1 Complexity

In 1971 Cook published the famous result on the NP-completeness of Satisfiability [11]. In 1978 Schaeffer introduced the Generalized Satisfiability Problem[24] and proved a dichotomy theorem which states that if P≠NP then all Generalized Satisfiability Problems are either in P or NP-complete.

Given a set of boolean functions $S = \{f_1, \ldots, f_k\}$ Schaeffer defined an *S-formula* as a conjunction of constraints of the form $f[x_{j_1}, \ldots, x_{j_l}]$, where f is a member of S, l is the arity of f and for $i = 1, \ldots, l$, x_{j_i} is a boolean variable. He defined the *S-Satisfiability problem* (also referred to as *the generalized satisfiability problem*) as deciding if a given S-formula is satisfiable.

Schaeffer also investigated the complexity of generalized satisfiability and proved the following:

Theorem 2.1 *[24] If P≠NP, S-Satisfiability is in P if and only if one of the following sentences is true. In all other cases S-Satisfiability is NP-complete.*

1. *Every function in S is 1-valid (0-valid).*

2. *Every function in S is weakly positive (weakly negative).*

3. *Every function in S is logically equivalent to a CNF formula having all clauses of size at most two.*

4. *Every function in S is logically equivalent to a system of linear equations over the boolean domain.*

Observation 2.8 Because $BCSP_{n,m}(\mathcal{F})$ uses random sign vectors, whp it is not 1-valid or 0-valid and it is not weakly positive or negative. If the maximum arity is at least 3 and the functions are globally consistent not all functions in $\mathcal{F}$ are logically equivalent to a 2-SAT formula. Therefore, by Schaeffer's result, if $\mathcal{F}$ consists of GCBC, unless all functions in F are logically equivalent to a system of linear equations, $BCSP_{n,m}(\mathcal{F})$ is NP-complete.

2.3.2 Random Models and Their Properties

Given $k \in \mathbb{N}^*$, $n \in \mathbb{N}^*$, $m \in \mathbb{N}$, and a set $\{x_1, \ldots, x_n\}$ of boolean variables, the k-SAT$_{n,m}$ model[2] generates random SAT instances with n variables consisting of m clauses of size k. Each clause C_i is built by the following procedure:

- select an ordered set of k distinct variables $(x_1^i, \ldots, x_k^i)$;
- select a random sign vector τ;
- let C_i be $\vee_{id_k}^{\tau}[x_1^i, \ldots, x_k^i]$.

For $k \leq 2$, k-SAT is solvable in polynomial time; for $k \geq 3$ k-SAT is NP-complete[11, 19]. In the early nineties it was empirically noticed [8, 20] that for fixed n, as m increases, the probability that an instance of random k-SAT being satisfiable drops sharply from 1 to 0 (see Fig. 2.1). The phenomenon has been called *the phase transition (PT)*. It was also noticed that the ratio of clauses to variables (m/n) at which the transition takes place seems to depend only on k (for large values of n). The ratio has been called *the threshold (value)* or *the cross over point*. Associated with the PT is an easy-hard-easy pattern of the instances difficulty

[2]When no confusion arises we will use random k-SAT to refer to k-SAT$_{n,m}$.

(as measured by the running time of complete solvers), the hardest instances being clustered around the threshold (see Fig.2.2). It has been conjectured that for any $k \geq 2$ there exists a value c_k such that for any $\epsilon > 0$

$$\lim_{n\to\infty} \mathbf{Pr}\left[k\text{-SAT}_{n,(c_k-\epsilon)n} \text{ is satisfiable}\right] = 1$$
$$\lim_{n\to\infty} \mathbf{Pr}\left[k\text{-SAT}_{n,(c_k+\epsilon)n} \text{ is satisfiable}\right] = 0.$$

For $k = 2$ it was proved [10, 17, 15] that $c_k = 1$. For $k \geq 3$ there is no such proof. However, in [16] Friedgut proved a weaker result which implies that for any k and n there exists a value $c_{k,n}$ such that for any $\epsilon > 0$

$$\lim_{n\to\infty} \mathbf{Pr}\left[k\text{-SAT}_{n,(c_{k,n}-\epsilon)n} \text{ is satisfiable}\right] = 1$$
$$\lim_{n\to\infty} \mathbf{Pr}\left[k\text{-SAT}_{n,(c_{k,n}+\epsilon)n} \text{ is satisfiable}\right] = 0.$$

Friedgut studied the $G_{n,p}$ model of a random graph, which generates a graph with n vertices. Each of the possible $\binom{n}{2}$ edges is added with probability p. He considered a monotone graph property $\mathcal{P}$. A graph property is *monotone* if any supergraph of a graph having $\mathcal{P}$ has $\mathcal{P}$. For $\epsilon > 0$ define p_0 such that $\mathbf{Pr}[G_{n,p_0} \text{ has } \mathcal{P}] = \epsilon$, p_1 such that $\mathbf{Pr}[G_{n,p_1} \text{ has } \mathcal{P}] = 1 - \epsilon$ and p_c such that $\mathbf{Pr}[G_{n,p_c} \text{ has } \mathcal{P}] = 1/2$. $p_1 - p_0$ is referred to as *the threshold length* of $\mathcal{P}$ and p_c is referred to as *the critical point* of $\mathcal{P}$. $\mathcal{P}$ is said to have a *sharp threshold* if $\lim_{n\to\infty} \frac{p_1-p_0}{p_c} = 0$. If $\frac{p_1-p_0}{p_c}$ is bounded away from 0 for every n, then $\mathcal{P}$ has a coarse threshold.

In [16] Friedgut proved that if a monotonic graph property $\mathcal{P}$ does not have a sharp threshold then it can be arbitrarily closely approximated by the property of having a subgraph from a given (finite, independent of n)-list of balanced graphs.

Observation 2.9 If a problem exhibits a sharp threshold, then any lower bound for unsatisfiability is a lower bound of the threshold, and any upper bound for satisfiability is an upper bound for the threshold.

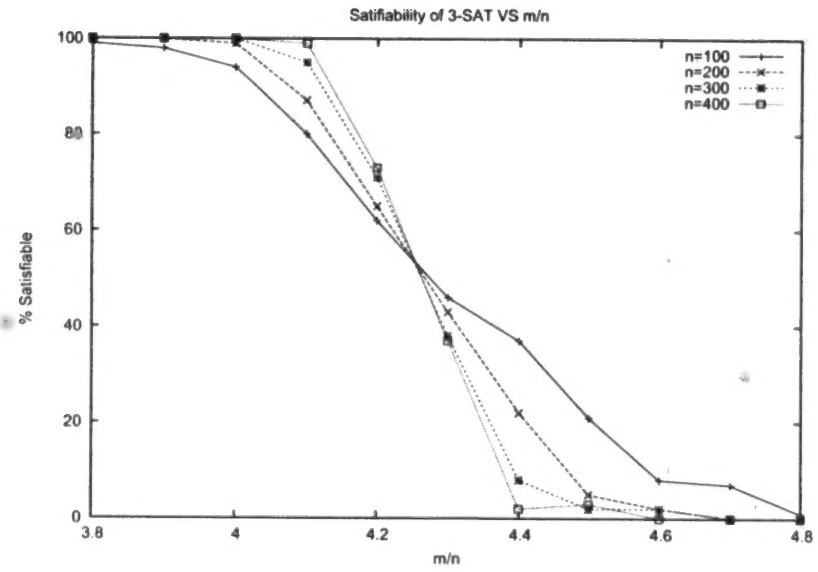

Figure 2.1: Phase transition of 3-SAT

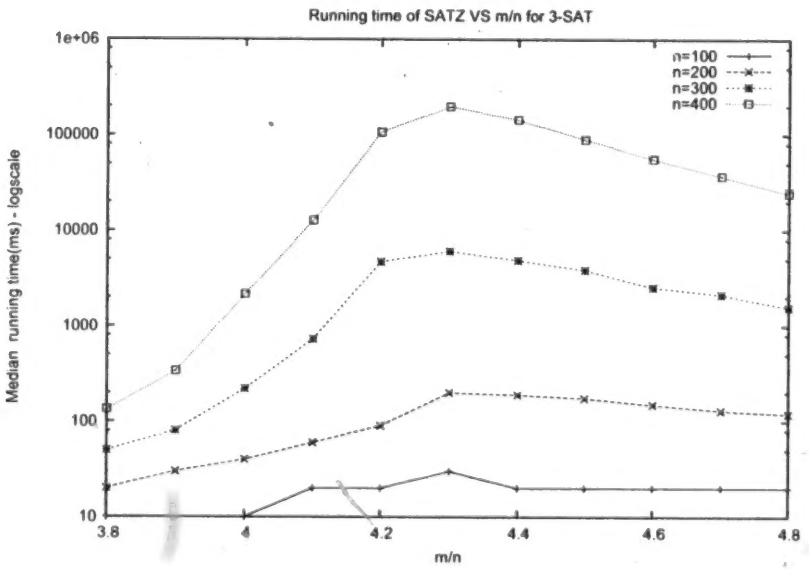

Figure 2.2: The easy-hard-easy pattern for 3-SAT

Besides a sharp threshold proof for random k-SAT, Friedgut's paper offered a methodology for proving the sharp threshold property for other combinatorial problems. As usual, generalized satisfiability was among the first ones to which it was applied.

Molloy [21] introduced a general model for random CSP and studied its threshold behavior. He considered uniform CSP's (in which each variable takes the same set of possible values). For the purposes of this thesis we will restrict the possible set of values to the boolean domain. Given $k \geq 2$, $n \geq 1$, a set of boolean variables $\{x_1, \ldots, x_n\}$ and a probability distribution $\mathcal{P}$ over the set of all nonempty boolean functions of arity k, the $CSP_{n,m}(\mathcal{P})$ model generates a CSP with m constraints as follows:

- generate a random k-uniform hypergraph with m hyperedges;
- for each hyperedge randomly permute the variables it connects, select a constraint function according to $\mathcal{P}$ and create a constraint by applying f to the set of permuted variables.

Molloy proved that the threshold behavior depends on the support of $\mathcal{P}$ - the set of functions to which $\mathcal{P}$ assigns positive probabilities.

Molloy used the following definition of a *sharp threshold*: $CSP_{n,m}(\mathcal{P})$ "is said to have a sharp threshold of satisfiability if there exists a function $c(n)$ bounded away from 0 such that for any $\epsilon > 0$, if $m < (c(n) - \epsilon)n$ then $CSP_{n,m}(\mathcal{P})$ is whp satisfiable and if $m > (c(n) + \epsilon)n$ then $CSP_{n,m}(\mathcal{P})$ is whp unsatisfiable".[3] He gave sufficient conditions for the existence of a sharp threshold.

A value a, such that $\lim_{n\to\infty} \mathbf{Pr}\left[CSP_{n,an}(\mathcal{P}) \text{ is satisfiable}\right] > 0$ is called *a lower bound for unsatisfiability*. A value a, such that $\lim_{n\to\infty} \mathbf{Pr}\left[CSP_{n,an}(\mathcal{P}) \text{ is satisfiable}\right] =$

[3] As the satisfiability transition of $CSP_{n,m}(\mathcal{P})$ happens for m varying linearly with n [21] this notion of sharp threshold is equivalent to the one introduced by Friedgut[16].

0 is called *an upper bound for satisfiability.*

Under certain assumptions on $\mathcal{P}$ he gave a coarse lower bound for unsatisfiability of $CSP_{n,m}(\mathcal{P})$, based on ensuring that the random hypergraph is acyclic. He also gave an upper bound for satisfiability using the first moment method. In the context of BCSP his results imply that:

- if the support of $\mathcal{P}$, denoted by $\mathcal{S}(\mathcal{P})$, has the property that for any $f \in \mathcal{S}(\mathcal{P})$ there exists $g \in \mathcal{S}(\mathcal{P})$ such that for any $\{x_1, \ldots, x_k\} \in \{0,1\}^k$ $f(x_1, \ldots, x_k) = g(1 - x_1, \ldots, 1 - x_k)$, and
- at least one of the functions in the support of $\mathcal{P}$ is not 1-valid, and
- at least one of the functions in the support of $\mathcal{P}$ does not depend on one literal,

then $\frac{1}{k(k-1)}$ is a lower bound for unsatisfiability and $2^k \ln k/A$ is an upper bound for satisfiability of $CSP_{n,m}(\mathcal{P})$, where $A = \min_{\mathcal{P}(C)>0} \mathcal{P}(C)$.

Observation 2.10 If $\mathcal{F}$ consists of GCBC it has all three properties above, and therefore there exists $B(\mathcal{F})$, such that for any $c > B$, $BCSP_{n,cn}(\mathcal{F})$ is whp unsatisfiable.

Independent of Molloy's work, Creignou and Daudé[13] introduced the following model for *random symmetric boolean CSP's*: given $n \in \mathbb{N}^*$, $p \in [0,1]$ and a set of boolean functions $\mathcal{F}$ of arity k, $S^{\mathcal{F}}(n,p)$ has n variables $\{x_1, \ldots, x_n\}$, and for each ordered subset of k distinct variables $(x_1^i, \ldots, x_k^i)$ and each $f \in \tilde{\mathcal{F}}$, $f[x_1^i, \ldots, x_k^i]$ is added independently with probability p. The same authors proved that $S^{\mathcal{F}}(n,p)$ exhibits a transition in satisfiability. They obtained a coarse lower bound on the unsatisfiability based on the existence of a minimal unsatisfiable subformula. They also proved the following theorem which gives necessary and sufficient conditions for the existence of a sharp threshold:

Theorem 2.2 *[13] If $\mathcal{F}$ contains a function strongly depending on one literal or on a 2-XOR relation, $S^{\mathcal{F}}(n,p)$ exhibits a coarse threshold. In all other cases it has a sharp threshold.*

Observation 2.11 If $\mathcal{F}$ consists of GCBC and is such that for any $f \in F$, $P(f) = \frac{|\tilde{f}|}{\sum_{f \in F} |\tilde{f}|}$, i.e. the functions in $\tilde{\mathcal{F}}$ have equal probability, the previous theorem applies to $BCSP_{n,m}(\mathcal{F})$.

Building on Molloy's work, Istrate [18] gave more general sufficient conditions for the existence of sharp threshold of a slightly different model for random CSP's. Again we will present the model restricted to boolean CSP. Let $k \geq 2$, $p \in [0,1]$ and $\mathcal{C}$ be a subset of the nonempty boolean functions of arity k. The $CSP_{n,p}(\mathcal{C})$ has $n \in \mathbb{N}^*$ variables $\{x_1, \ldots, x_n\}$ and for each ordered set of k variables $(x_1^i, \ldots, x_k^i)$ and each $f \in \mathcal{C}$, $f[x_1^i, \ldots, x_k^i]$ is independently added with probability p to the instance. The result proved by Istrate for the general CSP, has the following:

Corollary 2.1 *[18] The random boolean CSP model $CSP_{n,p}(\mathcal{C})$ has a coarse threshold if and only if some functions in $\mathcal{C}$ strongly depend on one literal or on a 2-XOR relation. In all other cases it has a sharp threshold.*

In [23] Molloy and Salavatipour studied the *uniform constrainedness* model for CSP's, $\mathcal{F}_{n,M}^{d,k,t}$ for which necessary conditions for a sharp threshold are given in [22]. If the variables are restricted to take values from the boolean domain, $\mathcal{F}_{n,m}^{2,k,t}$ is defined as follows: Let $k \geq 2$, $n \in \mathbb{N}^*$, $t \in \mathbb{N}^*$, $M \in \mathbb{N}$ and $\{x_1, \ldots, x_n\}$ be a set of variables. Choose uniformly at random a k-uniform hypergraph with n vertices and M hyper-edges. For each edge choose a function f uniformly at random from the set of all boolean functions of arity k which have exactly t nogoods and add the constraint generated by applying f to the ordered set of vertices connected by the edge. The authors gave a complete characterization of the resolution complexity

of $\mathcal{F}_{n,M}^{d,k,t}$. They also established an upper bound for the satisfiability of $\mathcal{F}_{n,M}^{d,k,t}$. For $\mathcal{F}_{n,m}^{2,3,3}$ they proved a lower bound for unsatisfiability, which equals the upper bound for satisfiability. Thus they established the exact value of the satisfiability threshold for $\mathcal{F}_{n,m}^{2,3,3}$.

2.3.3 Lower Bounds for Unsatisfiability

In 1986, Chao and Franco [6] gave a lower bound on the unsatisfiability of random 3-SAT by analyzing Unit Clause (UC)[4] , a random algorithm for solving SAT which does not backtrack. They proved that for $m/n < 2.66$, random 3-SAT is wpp satisfiable. The novelty element of their proof was the use of differential equations for estimating the number of clauses of different lengths.

Following a similar idea and using the differential equation method introduced by Wormald[25], Achlioptas proved several lower bounds for random k-SAT [1, 4]. Given the sharp threshold behavior of random k-SAT any lower bound for unsatisfiability is also a lower bound for the threshold.

We will present a theorem by Wormald [25] which is the foundation of the differential equations method.

Let $k \in \mathbb{N}^*$, for $i = 1, \ldots, k$ let Y_i be a discrete time random process, $\mathbf{H}(t)$ be the history of the processes up to time t, $\mathbf{H}(t) = \vec{Y}(0), \ldots, \vec{Y}(t)$, where for any time t, $\vec{Y}(t) = (Y_1(t), \ldots, Y_k(t))$.

The asymptotic notations o and O are for $n \to \infty$, but uniform over all the other variables. "Uniformly" is used to refer to the convergence implicit in the $o()$ terms. We say that a random variable $X = o(f(n))$ always if $\max\{x\colon \mathbf{Pr}[X = x] \neq 0\} = o(f(n))$. A function $f : \mathbb{R}^i \to \mathbb{R}$ satisfies a *Lipschitz condition* on a domain $D \subset \mathbb{R}^i$ if there exists a constant L s.t. $|f(x_1, \ldots, x_i) - f(y_1, \ldots, y_i)| \leq L \sum_{j=1}^{i} |y_j - x_j|$.

[4]UC is formally presented in Fig. 2.3

Theorem 2.3 *(Wormald's [25]) Let $k \in \mathbb{N}^*$, Y_i, $i = 1, \ldots, k$ be discrete time real valued random processes, $B > 0$ s.t. $Y_i(t) \leq Bn$ for all i, n and t, $\mathbf{H}(t)$ be the history of the processes up to time t, $\mathcal{D}$ be some bounded, connected, open set containing the closure of:*

$$\{(0, y_1, \ldots, y_k) \colon \mathbf{Pr}\left[Y_i(0) = y_i n, i = 1, \ldots, k\right] \neq 0 \text{ for some } n.\}$$

Let $f_i : \mathbb{R}^k \to \mathbb{R}$, $i = 1, \ldots, k$, and suppose that for some $M = M(n)$ the following relations are true:

(i) for all i and uniformly over all $t < M$

$$\mathbf{E}\left(Y_i(t+1) - Y_i(t) | \mathbf{H}(t)\right) = f_i(t/n, Y_1(t)/n, \ldots, Y_k(t)/n) + o(1);$$

(ii) for all i and uniformly over all $t < M$

$$\mathbf{Pr}\left[|Y_i(t+1) - Y_i(t)| > n^{1/5} \middle| \mathbf{H}(t)\right] = o(n^{-3});$$

(iii) for each i the function f_i, is continuous and satisfies a Lipschitz condition on $\mathcal{D}$.

Then:

(a) for $(0, \hat{z}_1, \ldots, \hat{z}_k) \in \mathcal{D}$ the system of differential equations

$$\frac{dz_i}{ds} = f_i(s, z_1, \ldots, z_k), i = 1, \ldots, k$$

has a unique solution in $\mathcal{D}$, for $z_i : \mathbb{R} \to \mathbb{R}$, with an initial condition $z_i(0) = \hat{z}_i$. The solution extends to points arbitrarily close to the boundary of $\mathcal{D}$;

(b) For each i, whp $Y_i(t) = z_i(t/n)n + o(n)$, uniformly for $0 \leq t \leq \min(\alpha n, M)$, where $z_i(s)$ is the solution of (a), with $\hat{z}_i = Y_i(0)/n$, and $\alpha = \alpha(n)$ is the supremum of those s to which the solution can be extended.

One of the notes in [25] states that the theorem remains valid if in (i) and (ii) "always" is replaced by the event that $(t/n, Y_1(t)/n, \ldots, Y_k(t/n)) \in \mathcal{D}$, and in the conclusion (b) the upper bound on s is replaced by the first s for which the event fails.

In the rest of this chapter we will show how, using Wormald's Theorem, a lower bound for unsatisfiability of $BCSP_{n,m}(\mathcal{F})$ can be obtained for the case where $\mathcal{F}$ consists of GCBC. The analysis is based on the performance of UC. This corresponds to extending the lower bound result from SAT to BCSP.

2.4 A lower bound for unsatisfiability

In this section we will prove the following:

Theorem 2.4 *For any family $\mathcal{F}$ of distinct GCBC functions of arity at least 3 and any $\delta > 0$ if*

$$m < \left(\frac{1}{2T_{\mathcal{F}}} - \delta\right) n, \tag{2.1}$$

then $BCSP_{n,m}(\mathcal{F})$ is asymptotically satisfiable wpp, where $T_{\mathcal{F}}$ is the maximum[5] over $x \in [0,1]$ of $h_{\mathcal{F}}(x) = (1-x)\sum_{f\in F} ar(f)(ar(f)-1)x^{ar(f)-2}P(f)\kappa(f)$. If $BCSP_{n,m}(\mathcal{F})$ has a sharp threshold ($\mathcal{F}$ is as in Observation 2.11) then $BCSP_{n,m}(\mathcal{F})$ is asymptotically satisfiable whp.

The result is inspired by a similar theorem presented by Achlioptas[2] for random k-SAT, which is based on the differential equations method introduced by Wormald[25].

Corollary 2.2 *If all functions in $\mathcal{F}$ have arity k, then $BCSP_{n,m}(\mathcal{F})$ is satisfiable wpp for $m < \left(\frac{1}{\mathcal{P}(\mathcal{F})}\frac{1}{2k}\left(\frac{k-1}{k-2}\right)^{k-2} - \delta\right) n$, where $\mathcal{P}(\mathcal{F}) = \sum_{f\in F} P(f)\kappa(f)$.*

[5] As $h_{\mathcal{F}}(x)$ is the product of a positive number and a polynomial function with positive coefficients, $T_{\mathcal{F}} > 0$.

Proof: In this case, $h_{\mathcal{F}}(x) = k(k-1)(1-x)x^{k-2}\sum_{f\in F} P(f)\kappa(f)$. By analyzing the derivative of $h_{\mathcal{F}}(x)$, on $[0,1]$ we get that $h_{\mathcal{F}}$ reaches its maximum over $[0,1]$, when $x = \frac{k-2}{k-1}$, and the maximum is $T = k\sum_{f\in F} P(f)\kappa(f)\left(\frac{k-2}{k-1}\right)^{k-2}$, which implies the result. □

As particular cases we get a lower bound of $\frac{2^{k-2}}{k}\left(\frac{k-1}{k-2}\right)^{k-2}$ for the threshold of random k-NAE-SAT. For random k-SAT we get a lower bound of $\frac{2^{k-1}}{k}\left(\frac{k-1}{k-2}\right)^{k-2}$ which agrees with the result in [2].

Corollary 2.3 *For $M < (\frac{1}{2T} - \delta)n$, $\mathcal{F}^{2,k,2}_{n,M}$ is whp satisfiable, where T is the maximum over $x = [0,1]$ of the function*

$$h(x) = \frac{k(k-1)}{2^{2k-2}}(1-x)x^{k-3}\left(x(2^{k-1}-k)+k-2\right)$$

Proof: In $\mathcal{F}^{d,k,t}_{n,M}$, the constraints are created by randomly choosing t nogoods. In our case 2 different nogoods will be randomly chosen out of the 2^k possible ones. Thus there are are $\binom{2^k}{2}$ different ways of choosing 2 different nogoods for a constraint. If the 2 nogoods have at least 2 variables which appear with different signs then the constraint is GCBC. Otherwise the constraint is equivalent to a constraint of arity $k-1$ with one randomly chosen nogood. It follows that F will consist of a function of arity $k-1$ with one no good and a set of functions of arity k with two nogoods. All functions are GCBC and their constrainedness is $\frac{1}{2^{k-1}}$. As there are $k2^{k-1}$ ways of choosing such a constraint, the function of arity $k-1$ is chosen with probability $\frac{k}{2^{k-1}}$. Thus by applying Theorem 2.4 we get the result. □

Observation 2.12 We can relax the condition on the arity of functions in F, by allowing functions of arity 2. All GCBC functions of arity 2 are equivalent (via permutations of variables and signs) with 2-SAT or with 2-XOR. To preserve the conditions for the existence of the sharp threshold, we must eliminate 2-XOR from F. The theorem remains true.

The proof of Theorem 2.4 shows that the Unit Clause algorithm (UC) succeeds with positive probability when applied to an instance satisfying the hypothesis. If $BCSP_{n,m}(\mathcal{F})$ has a sharp threshold ($\mathcal{F}$ as in Observation 2.11) this implies whp. The proof is obtained by the following steps:

- Define the process graph associated with a family of distinct GCBC and list some of its properties. The process graph is a directed acyclic weighted graph, used to keep track of the classes of constraints resulting from application of UC to $BCSP_{n,m}(\mathcal{F})$;
- Show that the random processes which count the number of constraints in each class of the process graph meet the requirements of Wormald's theorem;
- Apply Wormald's Theorem, solve the resulting system of differential equations, and find "a good" approximation of the density of constraints of arity 2;
- Show that if during the evolution of UC the density of constraints of arity 2 is whp bounded away from 1, then wpp UC will succeed on $BCSP_{n,m}(\mathcal{F})$ as in the hypothesis;
- Impose the inequality on the density of constraints of arity 2 to obtain the desired lower bound.

A *unit constraint(clause)* is a constraint(clause) of arity 1. The Unit Clause algorithm (see Fig. 2.3) tries to satisfy unit clauses if they exist. At each step it chooses one unassigned literal, b, which it sets to 1, and then it simplifies the instance by:

- Discarding all constraints which are made trivial by the assignment;
- Removing the negation of the chosen literal from all constraints containing it, i.e. replace the constraints containing $var(b)$, by the corresponding constraints reduced by the assignment.

If the instance contains unit constraints, one of them is chosen at random and its literal is satisfied. Otherwise the literal to be satisfied is chosen at random from the set of remaining literals.

We will use the process graph to guide UC as well as to keep track of the classes of constraints at different steps of the algorithm. The process graph defines all nontrivial functions which can result from the original family of functions under all possible series of variable instantiations. The weight of an arc represents the probability that a constraint containing the literal chosen by UC moves along that arc.

2.5 Proof of the result

2.5.1 The process graph and its properties

In this subsection we present a formal definition of the process graph. We also mention its main properties which will be used in the proof of Theorem 2.4.

Let $\mathcal{F}$ be a family of distinct GCBC and $\mathbf{K} = \max_{f \in F} ar(f)$. The *process graph* associated with $\mathcal{F}$ is a weighted directed acyclic graph $\mathcal{G}(\mathcal{F}) = (\mathcal{V}, \mathcal{E}, \Psi)$. Where

- $\mathcal{V}$ is the set of vertices of $\mathcal{G}$, defined by the following rules[6]:

  (1) $\mathcal{V}_0 = \tilde{F}$

  (2) $\mathcal{V}_i = \{\tilde{f} | \tilde{f} \cap \mathbb{T} = \emptyset, \exists(\tilde{h} \in \mathcal{V}_{i-1}, h \in \tilde{h}, (j,e) \in AR(h) \times \{0,1\})$ s.t. $\prod_j^e(h) \in \tilde{f}\}$

  (3) $\mathcal{V} = \bigcup \mathcal{V}_i$.

- $\mathcal{E}$ is the arc set; $\mathcal{E} = \{\langle \tilde{f}, \tilde{g} \rangle | \tilde{f} \in \mathcal{V}_i, \tilde{g} \in \mathcal{V}_{i+1}, \exists(f \in \tilde{f}, (j,e) \in AR(f) \times \{0,1\})$ s.t. $\prod_j^e(f) \in \tilde{g}\}$;

[6] $\mathcal{V}$ is the closure of $\tilde{F}$ under $\prod_i^e$, excluding $\mathbb{T}$.

- $\Psi : \mathcal{V} \to \mathbb{Q}$ is the weight function defined as follows:
  for any edge $\langle \tilde{f}, \tilde{g} \rangle \in \mathcal{E}$, choose $f_1 \in \tilde{f}$
  let $\Psi(\langle \tilde{f}, \tilde{g} \rangle) = |\{(i,e) \in AR(f) \times \{0,1\} \colon \prod_i^e(f_1) \in \tilde{g}\}|/2ar(f)$.

To simplify the notation when no confusion arises, we will use f to refer to both f and $\tilde{f}$.

For each $f \in \mathcal{V}$ we define $IN(f) = \{g | \langle g, f \rangle \in \mathcal{E}\}$ and $OUT(f) = \{g | \langle f, g \rangle \in \mathcal{E}\}$.

Let U denote the boolean unit function $U(x) = x$, and E denote the empty function $E = \emptyset$.

Observation 2.13 The following relations are obvious:

- $\langle f, g \rangle \in \mathcal{E} \Rightarrow ar(f) = ar(g) + 1$. This implies that the process graph is a level graph;
- $g \in IN(f)$, $f = \mathcal{V} \setminus F \Rightarrow ar(g) = ar(f) + 1$;
- $g \in OUT(f)$, $f = \mathcal{V} \setminus \{E\} \Rightarrow ar(g) = ar(f) - 1$.

The proof of the following lemma follows easily from Observations 2.5, 2.7 and 2.13.

Lemma 2.1 *The process graph of a family of distinct GCBC of maximum arity* **K** *has the following properties:*

- *It is well defined;*
- *It is a weighted directed acyclic graph;*
- $\{f | f \in \mathcal{V}, ar(f) = 1\} = \tilde{U}$; *extending the notation we will use* U *to refer to* $\tilde{U}$;[7]
- $\{f | f \in \mathcal{V}, ar(f) = 0\} = \tilde{E} = E$;

[7] The only nonempty, nontrivial boolean functions of arity 1, are $U(x) = x$, and $g(x) = 1 - x$.

- $OUT(U) = E$, $IN(E) = U$;
- *All vertices except E and U are connected to U by a path;*
- *For each $f \in \mathcal{V}$, $1 \leq |OUT(f)| \leq 2ar(f)$ and* $1/2 \leq \sum_{g \in OUT(f)} \Psi(\langle f, g \rangle) \leq 1$;
- $|\mathcal{V}| \leq |F| 2^{\mathbf{K}} \mathbf{K}!$;
- *For any $f \in \mathcal{V} \setminus \{E, U\}$ and any $f' \in f$ f' is GCBC.*

For any $f \in \mathcal{V}$, and any $g \in \mathcal{V}$, we define $\mathbf{Paths}(f, g)$ to be the set of directed paths connecting f to g. For each directed path $P = (f, l_1, \ldots, l_i, g)$ connecting f to g we define $\mathbf{len}(P)$ as $\Psi(\langle f, l_1 \rangle) \times \cdots \times \Psi(\langle l_i, g \rangle)$. We use π_g^f to denote $\sum_{P \in \mathbf{Paths}(f,g)} \mathbf{len}(P)$. By convention we take $\pi_f^f = 1$ and $\pi_g^f = 0$ if there is no directed path from f to g.

Observation 2.14 By induction we have that for any $(f \neq g) \in \mathcal{V}^2$

- $\pi_g^f = \sum_{h \in OUT(f)} \Psi(\langle f, h \rangle) \pi_g^h$;
- $\pi_g^f = \sum_{h \in IN(g)} \Psi(\langle h, g \rangle) \pi_h^f$.

Lemma 2.2 *For any $f \in \mathcal{V}$, $\pi_U^f = 2\kappa(f)$.*

Proof: We will prove the lemma by induction on the arity of f.

Base step: For $ar(f) = 1$ as U is the only function of arity 1 in $\mathcal{V}$, the relation is true as $\pi_U^U = 1$, and $\kappa(U) = 1/2$.

Induction step: Suppose the relation is true for $ar(f) = j$. We will prove it is also true for $ar(f) = j + 1$. By Observation 2.14 $\pi_U^f = \sum_{g \in OUT(f)} \Psi(\langle f, g \rangle) \pi_U^g$. From

Lemma 2.1 it follows that for all $g \in OUT(f)$, g has arity j. Thus, applying the induction hypothesis we get

$$\pi_U^f = \sum_{g \in OUT(f)} \Psi(\langle f, g \rangle) 2\kappa(g) = 2 \sum_{g \in OUT(f)} \Psi(\langle f, g \rangle) ng(g)/2^{ar(g)}$$
$$= 2 \sum_{g \in OUT(f)} \frac{|\{(i,e) \in AR(f) \times \{0,1\} \colon \prod_i^e(f) \in g\}|}{2ar(f)} ng(g)/2^{ar(g)}.$$

Notice that $ng(\prod_i^e(f))$ is either 0 (if $\prod_i^e(f)$ is trivial), or strictly positive (if there exists an arc in $\mathcal{G}$, joining f to $\prod_i^e(f)$). Therefore:
$\pi_U^f = 2\frac{\sum_{i \in AR(f)} \sum_{e \in \{0,1\}} ng(\prod_i^e(f))}{ar(f)2^{ar(f)}} = 2 \sum_{i \in AR(f)} \frac{ng(\prod_i^0(f)) + ng(\prod_i^1(f))}{ar(f)2^{ar(f)}}$.
By Observation 2.5 $\pi_U^f = 2 \sum_{i \in AR(f)} \frac{ng(f)}{ar(f)2^{ar(f)}} = 2\kappa(f)$ □.

Observation 2.15 For any $f \in \mathcal{V}$ and any $g \in \mathcal{V}$, $\pi_g^f \leq 1$.

2.5.2 Proof of Theorem 2.4

We will show that UC, which is presented in Figure 2.3, succeeds wpp on any instance of $BCSP_{n,m}(\mathcal{F})$ satisfying the hypothesis.

We will use the following notations, some of which have been introduced before:

- $\kappa(f) = \frac{ng(f)}{2^{ar(f)}}$ to denote the density of nogoods of a function $f \in F$;
- $var(b)$ to denote the underlying variable of a literal b;
- "at time t" to denote after t steps of UC;
- $\nu(t)$ to denote the set of unassigned literals at time t;
- $C_f(t)$ to denote the multiset of constraints in f at time t, $c_f(t) = |C_f(t)|$, for each $f \in \mathcal{V}$;[8]

[8] As $C_f(t)$ is a multiset, in $c_f(t)$ we count each constraint as many times as it appears in $C_f(t)$

- $\hat{c}_f = c_f(0)/m$ to denote the initial fraction of constraints in C_f;
- $\Delta c_f(t) = c_f(t+1) - c_f(t)$ to denote the variation of c_f from time t to time $t+1$, for each $f \in \mathcal{V} \setminus \{U, E\}$;
- $\Delta c_U^+(t)$ to denote the number of unit constraints generated at time t;
- $\mathbf{K} = \max_{f \in F} ar(f)$ to denote the maximum arity of the functions in F;
- $v = |\mathcal{V}| - 2$ to denote the cardinality of $\mathcal{V} \setminus \{E, U\}$;
- $W : \{1, \ldots, v\} \to \mathcal{V} \setminus \{U, E\}$ to denote an indexing function of $\mathcal{V} \setminus \{E, U\}$;
- $\vec{V}(t) =< C_{W(1)}(t), \ldots, C_{W(v)}(t) >$ to denote the vector describing the number of constraints, in each class $C_{W(i)}$ at time t;
- $\mathbf{H}(t) =< \vec{V}(0), \ldots, \vec{V}(t) >$ to denote the vector of vectors ($(t+1) \times v$ matrix) that corresponds to the evolution of the number of constraints in the classes, up to time t;
- $\eta = |F|$ to denote the cardinality of F;
- $h_{\mathcal{F}}(x)$ to denote the function $(1-x) \sum_{f \in F} ar(f)(ar(f)-1)x^{ar(f)-2}P(f)\kappa(f)$;
- $T_{\mathcal{F}}$ to denote the maximum over $[0, 1]$ of $h_{\mathcal{F}}(x)$;
- $x_{\mathcal{F}} \in [0, 1]$ to denote the value for which $h_{\mathcal{F}}(x)$ takes its maximum over $[0, 1]$, $T_{\mathcal{F}} = h_{\mathcal{F}}(x_{\mathcal{F}})$.
- t_ε to denote $\lfloor (1-\varepsilon)n \rfloor$, for any $0 < \varepsilon < 1$;
- π_g^f to denote $\sum_{p \in \mathbf{Paths}(f,g)} \mathbf{len}(p)$, for $f \in \mathcal{V}$, $g \in \mathcal{V}$.

At the beginning of UC , $C_f(0) = \emptyset$ for each $f \in \mathcal{V} \setminus F$. For each $f \in F$, $C_f(0)$ has $\hat{c}_f m$ constraints. As UC proceeds, some constraints will be moved along the arcs of $\mathcal{G}$, and some constraints will be removed from the system. At the end of UC, $C_E(n)$ is the only class which may be nonempty, in which case, UC failed to find a solution. If at the end of UC $C_E(n)$ is empty, then a solution exists and it is represented by the set of literals chosen by UC. We will stop the UC procedure very close to its end , and we will prove that if at that time C_E and C_U are empty, then a solution can be easily found wpp[9]. Thus, if we can prove that wpp UC succeeds on a $BCSP_{n,m}(\mathcal{F})$, where F is a family of distinct GCBC, and m is as in Theorem 2.4, we get the desired lower bound.

Procedure **Reduce** checks if the constraints containing $var(b)$ are satisfied by setting b. If so, it simply discards them. Otherwise it eliminates b from them, getting constraints of smaller arity.

Observation 2.16 The process graph captures all of the information regarding the set of boolean functions on which the instances are generated. Thus **UC** and **Reduce** only have to deal with the tuples of literals on which the constraints are generated. This will ensure the independence of constraints within each class of constraints.

Observation 2.17 For an edge $\langle f, g \rangle \in \mathcal{E}$, $\Psi(\langle f, g \rangle)$ represents the probability that a constraint $C \in C_f$, containing the underlying variable of the chosen literal at time t will be moved to C_g after applying procedure **Reduce**.

The following lemma proves that the constraints in each class $C_f(t)$ are independent and uniformly distributed. Thus $C_f(t)$ is an instance of $BCSP_{n-t,c_f(t)}(f)$.

[9]The technique used in the proof does not work if UC runs until its end. This is why we stop UC close to its end.

```
Procedure UC(G, (C_f(0))_{f∈F})
{
  for t=1 to n do
    if C_U(t-1) ≠ ∅ then (forced step)
       pick b, uniformly at random from C_U(t-1)
    else (free step)
       pick uniformly at random any literal b ∈ ν(t-1)
    endif
    Reduce(G,b,t)
  end for
}
Procedure Reduce(G,b,t)
{
  ν(t) = ν(t-1) \ {b, ¬b}
  foreach f ∈ V do
    foreach C ⊊ C_f(t-1), s.t.  C contains var(b) do
      C_f(t) = C_f(t-1) \ {C}
      let C' be the constraint obtained by reducing b
       if C' ∉ T then
          let C_g be the class to which C' belongs
          C_g(t) = C_g(t-1) ∪ C'
      endif
    end foreach
  end foreach
}
```

Figure 2.3: The Unit Clause Algorithm

Lemma 2.3 *For each* $0 \leq t \leq n$, *and each* $f \in \mathcal{V} \setminus \{E\}$, *conditional on* $\mathbf{H}(t)$ *the constraints in* $C_f(t)$ *represent an instance of* $BCSP_{n-t,c_f(t)}(f)$, *i.e. they are independent and uniformly distributed.*

Proof: As UC is a myopic algorithm [2] the conclusion follows immediately. □

The following Lemma shows that if the expected number of unit clauses generated at any time up to t_ε is less then one, then at time t_ε both C_E and C_U will be empty wpp.

Lemma 2.4 *For* $\delta, \varepsilon > 0$, *if for all* $0 \leq t \leq t_\varepsilon$ *whp* $\mathbf{E}\left(\Delta c_U^+(t)\right) < 1 - \delta$, *then wpp* $C_U(t_\varepsilon) \cup C_E(t_\varepsilon) = \emptyset$, *where* $t_\varepsilon = (1-\varepsilon)n$.

Proof: $C_U(t)$ can be viewed as the queue of a server. As long as the incoming rate is whp less than the serving rate ($\mathbf{E}\left(\Delta c_U^+(t)\right) < 1 - \delta$), it is a stable server, for which the average (over time) length of the queue is less than a constant, let it be χ. Therefore, $\sum_{t=0}^{t_\varepsilon} C_U(t) \leq \chi n$. As the literals in $C_U(t)$ are independent and equally likely to be any literal in $\nu(t)$, the probability that the negation of the literal chosen at step t is not in $C_U(t)$(probability of not generating an empty clause at time t) is $\left(1 - \frac{1}{2(n-t)}\right)^{C_U(t)-1}$. Therefore the probability of not generating an empty clause up to time t_ε (probability of $C_E(t) = 0$) is larger than $\left(1 - \frac{1}{2\varepsilon n}\right)^{\chi n}$, which asymptotically equals $e^{-\chi/2\varepsilon} + o(1)$. Thus $C_E(t_\varepsilon) = 0$ wpp. As a stable server is at any time idle wpp, we have that $C_U(t_\varepsilon) = 0$ wpp, which ends the proof. □

Thus we need to focus on the expected number of unit clauses generated at each step.

As in [2] we get:

Lemma 2.5 *For all* $0 \leq t \leq n - \mathbf{K}$, *and* $f \in \mathcal{V} \setminus \{U, E\}$, *conditional on* $\mathbf{H}(t)$, *we have:*

$$\Delta c_f(t) = -B_f(t) + \sum_{h \in IN(f)} D_h^f(t); \tag{2.2}$$

where $B_f(t) \approx \mathbf{Bin}\left(c_f(t), \frac{ar(f)}{n-t}\right)$, $D_h^f(t) \approx \mathbf{Bin}\left(c_h(t), \frac{ar(h)\Psi(\langle h,f\rangle)}{(n-t)}\right)$, *and where* $Z \approx \mathbf{Bin}(N,p)$, *means* Z *is a random variable having a Binomial distribution with* N *trials and probability of success* p.

Proof: Let b be the literal chosen at time t. The negative quantities represent the number of constraints leaving each class. Var(b) must appear in any constraint which leaves a class f. As the constraints of $C_f(t)$ are equally likely to be any formula of that type over the $n-t$ unset variables, each variable has probability $ar(f)/(n-t)$ to appear in any constraint in $C_f(t)$. The $c_f(t)$ constraints in $C_f(t)$ are independent of each other, and thus the number of constraints in $C_f(t)$ which contain var(b) follows a binomial distribution with $c_f(t)$ trials, each with probability of success $ar(f)/(n-t)$. The positive quantities represent the number of constraints entering class f as a result of applying **Reduce** at time t. Let g, be any class from which constraints can flow into f. That means there must exist an arc in the process graph which joins h to f. If a constraint C is to enter f from h, then it must contain $var(b)$, which happens with probability $ar(h)/(n-t)$. By Observation 2.17 the probability that C, which contains $var(b)$, moves to f is $\Psi(\langle h,f\rangle)$. Thus the probability that a constraint in h moves to f is $ar(h)\Psi(\langle h,f\rangle)/(n-t)$. From the independence and uniform distribution of constraints in $C_h(t)$, we get that the event of C moving to f, follows a binomial distribution $\mathbf{Bin}\left(c_h(t), \frac{ar(h)\Psi(\langle h,f\rangle)}{(n-t)}\right)$. The result is obtained by noticing that the number of constraints entering f is the sum of the inputs from all classes having an end point in f. □

Taking expectation on (2.2) we get:

$$\mathbf{E}\left(\Delta c_f(t)|\mathbf{H}(t)\right) = -\frac{ar(f)}{n-t}c_f(t) + \frac{1}{n-t}\sum_{h\in IN(f)} ar(h)\Psi(\langle h,f\rangle)c_h(t), \qquad (2.3)$$

for each $f \in \mathcal{V} \setminus \{U,E\}$.

Equation (2.3) can be rewritten as:

$$\mathbf{E}\left(\Delta c_f(t)|\mathbf{H}(t)\right) = -\frac{ar(f)}{1-t/n}\frac{c_f(t)}{n} + \frac{1}{1-t/n}\sum_{h\in IN(f)} ar(h)\Psi(\langle h,f\rangle)\frac{c_h(t)}{n}. \quad (2.4)$$

Similar to Lemma 2.5 we have that conditional on $\mathbf{H}(t)$, $\Delta c_U^+(t) = \sum_{h\in IN(U)} D_h^U(t)$, where $D_h^f(t)$ is defined as in Lemma 2.5. It follows that conditional on $\mathbf{H}(t)$, the expected number of unit clauses generated at time t is

$$\mathbf{E}\left(\Delta c_U^+(t)\right) = \frac{1}{(n-t)}\sum_{h\in IN(U)} \Psi(\langle h,U\rangle)ar(h)c_h(t). \quad (2.5)$$

As usual [2] with this kind of result, at this point we will apply Wormald's Theorem [25] to get $o(n)$ approximations of $c_f(t)$ for $f \in \mathcal{V}\setminus\{E,U\}$.

The following lemma, proves that our system meets the conditions of the Wormald Theorem.

Lemma 2.6 *For any instance $BCSP_{n,m}(\mathcal{F})$ and any $\varepsilon > 0$, let*

- *$M = (1-\varepsilon)n$, $C = B(\mathcal{F})+\varepsilon$, where $B(\mathcal{F})$ is the upper bound from Observation 2.10;*
- *$Y_i(t) = c_{W(i)}(t)$, for each $i = 1,\ldots,v$;*
- *$f_i(t,z_1,\ldots,z_v) = -\frac{ar(W(i))}{1-t}z_i + \frac{1}{(1-t)}\sum_{h\in IN(W(i))} ar(h)\Psi(\langle h,W(i)\rangle)z_{W^{-1}(h)}$, for each $i = 1,\ldots,v$;*
- *$\mathcal{D} = \{(t,z_1,\ldots,z_v):\ -\varepsilon < t < 1\ , -\varepsilon < z_i < C$, for each $i = 1,\ldots,v\}$.*

The following relations, which represent the hypothesis of the Wormald Theorem, are true:

(i) for each $i = 1,\ldots,v$ and uniformly over all $t < M$ s.t.
$(t/n, Y_1(t)/n,\ldots,Y_v(t)/n) \in \mathcal{D}$

$$\mathbf{E}\left(Y_i(t+1) - Y_i(t)|\mathbf{H}(t)\right) = f_i(t/n, Y_1(t)/n,\ldots,Y_v(t)/n) + o(1);$$

(ii) for each $i = 1, \ldots, v$ and uniformly over all $t < M$ s.t.
$(t/n, Y_1(t)/n, \ldots, Y_v(t)/n) \in \mathcal{D}$

$$\mathbf{Pr}\left[|Y_i(t+1) - Y_i(t)| > n^{1/5} \middle| \mathbf{H}(t)\right] = o(n^{-3});$$

(iii) for each $i = 1, \ldots, v$ the function f_i, is continuous and satisfies a Lipschitz condition on $\mathcal{D}$.

Proof: Relation (i) follows from Equation (2.4), noticing that $(1 - \varepsilon)n < n - \mathbf{K}$.

Relation (ii) requires us to prove that for each $f \in \mathcal{V} \setminus \{U, E\}$, we have

$$\mathbf{Pr}\left[|\Delta c_f(t)| > n^{1/5} \middle| \mathbf{H}(t)\right] = o(n^{-3})$$

If $|\Delta c_f(t)| = \left| -B_f(t) + \sum_{h \in IN(f)} D_h^f(t) \right|$ is to be larger than $n^{1/5}$, then either $B_f(t)$, or one of $D_h^f(t)$ (let us denote it by $Q_f(t)$), must be larger than $n^{1/5}/\mu_f$, where $\mu_f = 1 + |IN(f)|$. This also implies $q_f(t) > 0$,

where $q_f(t) = \begin{cases} c_f(t), & \text{if } Q_f(t) = B_f(t) \\ c_h(t), & \text{otherwise} \end{cases}$

From $1 \leq q_f(t) \leq Cn$, $\varepsilon n \leq (n - t) \leq n$, and $\mathbf{E}(Q_f(t)) = \nu_f \frac{q_f(t)}{n-t}$ it follows that for any $1 \leq t \leq t_\varepsilon$, $\frac{\nu_f}{n} \leq \mathbf{E}(Q_f(t)) \leq \frac{\nu_f C}{\varepsilon}$,

where $\nu_f = \begin{cases} ar(f), & \text{if } Q_f(t) = C_f(t) \\ ar(h)\Psi(\langle h, f \rangle), & \text{otherwise} \end{cases}$

We will use the Chernoff bound[9] of the Binomial distribution which states that for a random variable $V \approx \mathbf{Bin}(n, p)$, and any $\xi > 0$

$$\mathbf{Pr}[V \geq (1 + \xi)\mathbf{E}(V)] \leq e^{-\xi^2 \mathbf{E}(V)/3}$$

$$\mathbf{Pr}[V \geq (1 - \xi)\mathbf{E}(V)] \leq e^{-\xi^2 \mathbf{E}(V)/2}$$

Setting $\xi = \frac{n^{1/5}}{\mu_f \mathbf{E}(Q_f(t))} - 1$ in the Chernoff bound, we get:

$$\begin{aligned}
\mathbf{Pr}\left[Q_f(t) > n^{1/5}/\mu_f\right] &\leq e^{-\left(\frac{n^{1/5}}{\mu_f \mathbf{E}(Q_f(t))}-1\right)^2 \mathbf{E}(Q_f(t))/3} \\
&\leq e^{-\frac{1}{3}\left(\frac{n^{2/5}}{(\mu_f)^2 \mathbf{E}(Q_f(t))} - \frac{2n^{1/5}}{\mu_f} + \mathbf{E}(Q_f(t))\right)} \\
&\leq e^{-\frac{1}{3}\left(\frac{n^{2/5}\varepsilon}{(\mu_f)^2 \nu_f C} - \frac{2n^{1/5}}{\mu_f} + \frac{\nu_f}{n}\right)}.
\end{aligned}$$

Condition (ii) is obtained by noticing that

$\mathbf{Pr}\left[|\Delta c_f(t)| > n^{1/5}\right] \leq \mathbf{Pr}\left[Q_f(t) > n^{1/5}/\mu_f\right]$

and

$e^{-\frac{1}{3}\left(\frac{n^{2/5}\varepsilon}{(\mu_f)^2 \nu_f C} - \frac{2n^{1/5}}{\mu_f} + \frac{\nu_f}{n}\right)} = o(n^{-3})$.

As the functions f_i are linear in each z_j $(j = 1, \ldots, v)$, they satisfy a Lipschitz condition on those variables. As $0 \leq t \leq 1 - \varepsilon$, $1/(1 - t) \leq 1/\varepsilon$, the functions f_i satisfy a Lipschitz condition on t, which implies that f_i satisfy a Lipschitz condition on $\mathcal{D}$, which takes care of relation (iii). □.

Lemma 2.6 allows us to apply Wormald's Theorem and thus we get that:

$$\text{for each } f \in \mathcal{V} \setminus \{U, E\} \text{ and } 0 \leq t \leq t_\varepsilon,\ c_f(t) = nS_f(t/n) + o(n), \tag{2.6}$$

where $S_f(x)$ are the solutions of:

$$\begin{aligned}
&\frac{dS_f}{dx} = -\frac{ar(f)}{1-x} S_f + \frac{\sum_{h \in IN(f)} ar(h)\Psi(\langle h, f\rangle) S_h}{1-x}, \\
&\text{with initial conditions } S_f(0) = \hat{c}_f \frac{m}{n}.
\end{aligned} \tag{2.7}$$

The following lemma is used to obtain the solutions of the differential equations system, and thus estimates for $c_f(t)$.

Lemma 2.7 *For any $\varepsilon > 0$, $f \in \mathcal{V} \setminus \{E, U\}$, and any $0 \leq t \leq t_\varepsilon$,*

$$c_f(t) = m(1 - t/n)^{ar(f)} \left(\sum_{j \in F}^{ar(j) \geq ar(f)} (t/n)^{ar(j) - ar(f)} \binom{ar(j)}{ar(f)} \hat{c}_j \pi_f^j \right) + o(n).$$

Proof: We will prove the result by solving (2.7). The solutions of (2.7) are of the form $S_f(x) = (1-x)^{ar(f)} d_f(x)$ where $d_f(x)$ are the solutions of $(d_f)'(x) = \frac{1}{(1-x)^{ar(f)+1}} \sum_{h \in IN(f)} ar(h)\Psi(\langle h, f\rangle) S_h(x)$, with the initial conditions $d_f(0) = \hat{c}_f \frac{m}{n}$.

We will prove by induction on $l = \mathbf{K} - ar(f)$, that

$$S_f(x) = m/n(1-x)^{ar(f)} \left(\sum_{j \in F}^{ar(j) \geq ar(f)} x^{ar(j)-ar(f)} \binom{ar(j)}{ar(f)} \hat{c}_j \pi_f^j \right).$$

Base step: For $l = 0$, we have $ar(f) = \mathbf{K}$. The only functions of arity $\mathbf{K}$ are $g \in F$. As for any such g, $IN(g) = \emptyset$, we have $(d_g)' = 0$, which makes $d_g(x) = \hat{c}_g \frac{m}{n}$. Thus $S_g = \hat{c}_g m/n(1-x)^{\mathbf{K}} = m/n(1-x)^{ar(g)} \sum_{f \in F}^{ar(f) \geq ar(g)} \hat{c}_g \pi_g^f$, which agrees with the hypothesis.

Induction step: Suppose the induction hypothesis is true for some $l < \mathbf{K} - 2$. We want to prove it is true for $l+1$. Let $f \in \mathcal{V} \setminus \{U, E\}$, s.t. $ar(f) = \mathbf{K} - l - 1$. We get $S_f = (1-x)^{\mathbf{K}-l-1} d_f(x)$, where $(d_f)'(x) = \frac{1}{(1-x)^{\mathbf{K}-l}} \sum_{h \in IN(f)} ar(h)\Psi(\langle h, f\rangle) S_h(x)$, and $d_f(0) = \hat{c}_f \frac{m}{n}$. Based on the induction hypothesis we get:

$$\begin{aligned}(d_f)'(x) =& \frac{1}{(1-x)^{\mathbf{K}-l}} \sum_{h \in IN(f)} \frac{m}{n} (1-x)^{ar(h)} ar(h)\Psi(\langle h, f\rangle) \\ & \times \left(\sum_{j \in F}^{ar(j) \geq ar(h)} x^{ar(j)-ar(h)} \binom{ar(j)}{ar(h)} \hat{c}_j \pi_h^j \right)\end{aligned}$$

As for any $h \in IN(f)$, $ar(h) = ar(f) + 1$, we have

$$\begin{aligned}(d_f)'(x) &= \frac{m}{n} \sum_{h \in IN(f)} ar(h)\Psi(\langle h, f\rangle) \left(\sum_{j \in F}^{ar(j) \geq ar(h)} x^{ar(j)-ar(f)-1} \binom{ar(j)}{ar(h)} \hat{c}_j \pi_h^j \right) \\ &= \frac{m}{n} \left(\sum_{j \in F}^{ar(j) \geq \mathbf{K}-l} \hat{c}_j x^{ar(j)-ar(f)-1} (ar(f)+1) \binom{ar(j)}{ar(f)+1} \sum_{h \in IN(f)} \Psi(\langle h, f\rangle) \pi_h^j \right) \\ &= \frac{m}{n} \left(\sum_{j \in F}^{ar(j) \geq \mathbf{K}-l} \hat{c}_j x^{ar(j)-ar(f)-1} \binom{ar(j)}{ar(f)} \pi_f^j (ar(j) - ar(f)) \right)\end{aligned}$$

As $ar(f) = \mathbf{K} - l - 1$ it follows that:

$$(d_f)'(x) = \frac{m}{n} \sum_{j \in F}^{ar(j)>ar(f)} \hat{c}_j x^{ar(j)-ar(f)-1} \binom{ar(j)}{ar(f)} (ar(j) - ar(f)) \pi_f^j$$

which implies

$$d_f(x) = \frac{m}{n} \sum_{j \in F}^{ar(j)>ar(f)} x^{ar(j)-ar(f)} \binom{ar(j)}{ar(f)} \hat{c}_j \pi_f^j + \Delta.$$

From $d_f(0) = \hat{c}_f m/n$ we get that $\Delta = \hat{c}_f m/n$. Hence,

$$d_f(x) = \frac{m}{n} \left(\sum_{j \in F}^{ar(j) \geq ar(f)} x^{ar(j)-ar(f)} \binom{ar(j)}{ar(f)} \hat{c}_j \pi_f^j \right),$$

and thus

$$S_f(x) = \frac{m}{n} (1-x)^{ar(f)} \left(\sum_{j \in F}^{ar(j) \geq ar(f)} x^{ar(j)-ar(f)} \hat{c}_j \pi_f^j \binom{ar(j)}{ar(f)} \right),$$

which proves the induction step. Replacing S_f in Equations (2.6) we get the result of the lemma. □

We will impose an inequality on m/n using Equation (2.5).

Lemma 2.8 *For any $\delta > 0$ if*

$$\max_{x \in [0,1]} (1-x) \sum_{f \in F} x^{ar(f)-2} ar(f)(ar(f)-1) \hat{c}_f \kappa(f) < \frac{1}{2} \frac{1}{m/n + \delta} \tag{2.8}$$

then for any $0 < \varepsilon < 1$, $C_U(t_\varepsilon) \cup C_E(t_\varepsilon) = \emptyset$ wpp.

Proof: From Equation (2.5) and Lemma 2.7 we get that for $0 \leq t \leq t_\varepsilon$,

$$\mathbf{E}\left(\Delta c_U^+(t)\right) = \frac{1}{n-t} \sum_{h \in IN(U)} \left(m(1-t/n)^{ar(h)} ar(h) \Psi(\langle h, U \rangle) \right.$$
$$\left. \times \left(\sum_{j \in F}^{ar(j) \geq ar(h)} (t/n)^{ar(j)-ar(h)} \hat{c}_j \pi_h^j \binom{ar(j)}{ar(h)} \right) + o(n) \right)$$

As any $h \in IN(U)$ has arity 2, and any $f \in F$ has arity at least 3 we get:

$$\begin{aligned}
\mathbf{E}\left(\Delta c_U^+(t)\right) &= \frac{m}{n}(1-t/n)\left(\sum_{j\in F} 2(t/n)^{ar(j)-2}\binom{ar(j)}{2}\hat{c}_j \sum_{h\in IN(U)} \Psi(\langle h,U\rangle)\pi_h^j\right) + o(1) \\
&= \frac{2m}{n}(1-t/n)\left(\sum_{j\in F}(t/n)^{ar(j)-2}\hat{c}_j\binom{ar(j)}{2}\pi_U^j\right) + o(1) \\
&= \frac{2m}{n}(1-t/n)\sum_{j\in F}\left((t/n)^{ar(j)-2}\hat{c}_j\binom{ar(j)}{2}2\kappa(j)\right) + o(1) \\
&= \frac{2m}{n}(1-t/n)\sum_{j\in F}(t/n)^{ar(j)-2}ar(j)(ar(j)-1)\hat{c}_j\kappa(j) + o(1)
\end{aligned}$$

Consider the function $h(x) = (1-x)\sum_{f\in F} x^{ar(f)-2}ar(f)(ar(f)-1)\hat{c}_f\kappa(f)$ over the domain $[0,1]$; $h(x) \geq 0$ for any $x \in [0,1]$. Let $T = \max_{x\in[0,1]} h(x)$. T must exist as h is continuous in x, and the maximum is taken over a compact set; it is also true that $T > 0$. By equation (2.8) for any $0 \leq t \leq 1$, $\mathbf{E}\left(\Delta c_U^+(t)\right) < 1 - \delta_1$. Therefore, by Lemma 2.4 wpp $C_U(t_\epsilon) \cup C_E(t_\epsilon) = \emptyset$. □

We will prove the following lemma:

Lemma 2.9 *For any family $\mathcal{F}$ of distinct GCBC, and any $\delta > 0$ if*

$$m < \left(\frac{1}{2T_{\mathcal{F}}} - \delta\right)n \tag{2.9}$$

then, for m large enough,

$$\max_{x\in[0,1]}\left((1-x)\sum_{f\in F} x^{ar(f)-2}ar(f)(ar(f)-1)\hat{c}_f\kappa(f)\right) < \frac{1}{2}\frac{1}{m/n+\delta}$$

wpp for any instance of $BCSP_{n,m}(\mathcal{F})$.

Proof: Let $A : \{1,\ldots,\eta\} \to F$ be an enumeration of F and for any $\epsilon > 0$, let **Ball**$((x_1,\ldots,x_k),\epsilon)$, be the k-dimensional ball of radius ϵ centered in $(x_1,\ldots,x_k)$.

From equation (2.9) it follows that there exists $\alpha > 0$ s.t. $T_{\mathcal{F}}+\alpha < \frac{1}{2}\frac{1}{m/n+\delta}$. Consider the function $Y : [0,1]^{\eta+1} \to \mathbb{R}$ defined by

$$Y(x, z_1, \ldots, z_\eta) = (1-x)\sum_{i=1}^{\eta} x^{ar(A(i))-2} ar(A(i))(ar(A(i))-1)\kappa(A(i))z_i.$$

Notice that $Y(x_{\mathcal{F}}, P_{A(1)}, \ldots, P_{A(\eta)}) = T_{\mathcal{F}}$, and hence $Y(x, P_{A(1)}, \ldots, P_{A(\eta)}) \leq T_{\mathcal{F}}$ for any $x \in [0,1]$. Y is continuous in all arguments and defined on a compact set, and therefore it is uniformly continuous. Hence, there exists $\gamma > 0$ s.t. for any $x \in [0,1]$ and any $(z_1, \ldots, z_\eta) \in \mathbf{Ball}((P_{A(1)}, \ldots, P_{A(\eta)}), \gamma)$,

$$Y(x, z_1, \ldots, z_\eta) < Y(x, P(A_1), \ldots, P(A_\eta)) + \alpha \leq T_{\mathcal{F}} + \alpha$$

Therefore, for any $(z_1, \ldots, z_\eta) \in \mathbf{Ball}((P_{A(1)}, \ldots, P_{A(\eta)}), \gamma)$,

$$\max_{x\in[0,1]} Y(x, z_1, \ldots, z_\eta) \leq T_{\mathcal{F}} + \alpha < \frac{1}{2}\frac{1}{m/n+\delta}.$$

Thus, any instance of $BCSP_{n,m}(\mathcal{F})$ having $(\hat{c}_{A(1)}, \ldots, \hat{c}_{A(\eta)}) \in D$ satisfies the conclusion of the Lemma 2.9, where $D = \mathbf{Ball}((P_{A(1)}, \ldots, P_{A(\eta)}), \gamma)$. We will prove that for m large enough, $(\hat{c}_{A(1)}, \ldots, \hat{c}_{A(\eta)}) \in D$ wpp[10] for $BCSP_{n,m}(\mathcal{F})$.

$$\begin{aligned}
&\mathbf{Pr}\left[(\hat{c}_{A(1)}, \ldots, \hat{c}_{A(\eta)}) \in D\right] = \mathbf{Pr}\left[\bigcap_{i=1}^{\eta}\left\{\hat{c}_{A(i)} \in (P(A(i)) - \gamma, P(A(i)) + \gamma)\right\}\right] \\
&= 1 - \mathbf{Pr}\left[\bigcup_{i=1}^{\eta}\left\{\hat{c}_{A(i)} \notin (P(A(i)) - \gamma, P(A(i)) + \gamma)\right\}\right] \\
&\geq 1 - \sum_{i=1}^{\eta} \mathbf{Pr}\left[\hat{c}_{A(i)} \notin (P(A(i)) - \gamma, P(A(i)) + \gamma)\right] \qquad (2.10) \\
&\geq 1 - \sum_{i=1}^{\eta} \mathbf{Pr}\left[c_{A(i)}(0) \notin (P(A(i))m - \gamma m, P(A(i))m + \gamma m)\right] \\
&\geq 1 - \sum_{i=1}^{\eta} \mathbf{Pr}\left[|c_{A(i)}(0) - P(A(i))m| \geq \gamma m\right]
\end{aligned}$$

[10] We actually prove that $(\hat{c}_{A(1)}, \ldots, \hat{c}_{A(\eta)}) \in D$ whp

From Observation 2.4 we know that $(c_{A(1)}(0),\ldots,c_{A(\eta)}(0))$ follows a multinomial distribution $\mathbf{Mn}(m,(P(A(1)),\ldots,P(A(\eta))))$. Therefore for each $i=1,\ldots,\eta$, $c_{A(i)}(0)$, follows a binomial distribution $\mathbf{Bin}(m,P(A(i)))$.

By Chernoff bounds[9] we get that $\mathbf{Pr}\left[|V-\mathbf{E}(V)|\geq\xi\mathbf{E}(V)\right]\leq 2e^{-\xi^2\mathbf{E}(V)/2}$. Taking $\xi_i=\gamma/P(A(i))$ for each $i=1,\ldots,\eta$, we have that

$$\mathbf{Pr}\left[|c_{A(i)}(0)-P(A(i))m|\geq\gamma m\right]\leq 2e^{-\gamma^2 m/2P(A(i))}\leq 2e^{-\gamma^2 m/2\beta}, \tag{2.11}$$

where $\beta=\max_{i=1}^{\eta}P(A(i))$.

Combining equations (2.10) and (2.11) we get:

$$\mathbf{Pr}\left[(\hat{c}_{A(1)},\ldots,\hat{c}_{A(\eta)})\in D\right]\geq 1-\sum_{i=1}^{\eta}2e^{-\gamma^2 m/2\beta}\geq 1-2\eta e^{-\gamma^2 m/2\beta}$$

For $m>2\beta\eta/\gamma^2$, we have $1-2\eta e^{-\gamma^2 m/2\beta}\geq 1-\frac{2\eta}{e^{\eta}}>0$, for any $\eta\geq 1$, which ends the proof of Lemma 2.9. □

Combining Lemmas 2.9 and 2.8 we get that if m/n is chosen as in Lemma 2.9, then for any $0<\varepsilon<1$, $C_U(t_\varepsilon)\cup C_E(t_\varepsilon)=\emptyset$ wpp.

The following lemma proves that if we choose ε small enough then, the instance we are left with at time t_ε is satisfiable wpp.

Lemma 2.10 *If m/n is chosen as in Lemma 2.9, and $\varepsilon<\frac{n}{m\mathbf{K}(\mathbf{K}-1)\eta(\mathbf{K}!)^2 2^{\mathbf{K}+1}}$ then wpp the instance we are left with at t_ε is satisfiable.*

Proof: $C_U(t_\varepsilon)\cup C_E(t_\varepsilon)=\emptyset$ wpp. Thus, by Lemma 2.7 the instance we are left with at time t_ε has a number of constraints

$$\begin{aligned}T(t_\varepsilon)&=\sum_{f\in\mathcal{V}\setminus\{E,U\}}\left(m(1-t_\varepsilon/n)^{ar(f)}\sum_{j\in F}^{ar(j)\geq ar(f)}(t_\varepsilon/n)^{ar(j)-ar(f)}\binom{ar(j)}{ar(f)}\hat{c}_j\pi_f^j+o(n)\right)\\&=n\sum_{f\in\mathcal{V}\setminus\{E,U\}}\left(\frac{m}{n}\varepsilon^{ar(f)}\sum_{j\in F}^{ar(j)\geq ar(f)}(1-\varepsilon)^{ar(j)-ar(f)}\binom{ar(j)}{ar(f)}\hat{c}_j\pi_f^j+o(1)\right)\end{aligned} \tag{2.12}$$

As at time t_ε, there will be $n - t_\varepsilon$ variables left, we get that the constraints to variables ratio at time t_ε is:

$$\frac{T(t_\varepsilon)}{(n-t_\varepsilon)} = \frac{n}{n-n(1-\varepsilon)} \times$$
$$\sum_{f\in\mathcal{V}\setminus\{E,U\}} \left(\frac{m}{n}\varepsilon^{ar(f)} \sum_{j\in F}^{ar(j)\geq ar(f)} (1-\varepsilon)^{ar(j)-ar(f)} \binom{ar(j)}{ar(f)} \hat{c}_j \pi_f^j + o(1) \right)$$
$$= \frac{m}{n}\varepsilon \sum_{f\in\mathcal{V}\setminus\{E,U\}} \left(\varepsilon^{ar(f)-2} \sum_{j\in F}^{ar(j)\geq ar(f)} (1-\varepsilon)^{ar(j)-ar(f)} \binom{ar(j)}{ar(f)} \hat{c}_j \pi_f^j \right) + o(1)$$
$$\leq \frac{m}{n}\varepsilon \sum_{f\in\mathcal{V}\setminus\{E,U\}} \left(\sum_{j\in F}^{ar(j)\geq ar(f)} \hat{c}_j \pi_f^j \binom{ar(j)}{ar(f)} \right) + o(1)$$

By Observation 2.15 it follows that:

$$\frac{T(t_\varepsilon)}{(n-t_\varepsilon)} \leq \frac{m}{n}\varepsilon \sum_{f\in\mathcal{V}\setminus\{E,U\}} \left(\sum_{j\in F}^{ar(j)\geq ar(f)} \hat{c}_j \mathbf{K}! \right) + o(1)$$
$$\leq \frac{m}{n}\varepsilon \mathbf{K}! \sum_{f\in\mathcal{V}\setminus\{E,U\}} \left(\sum_{j\in F}^{ar(j)\geq ar(f)} \hat{c}_j \right) + o(1) \leq \frac{m}{n}\varepsilon \mathbf{K}! \sum_{f\in\mathcal{V}\setminus\{E,U\}} (1) + o(1).$$

By Lemma 2.1 $\frac{T(t_\varepsilon)}{(n-t_\varepsilon)} \leq m/n\varepsilon \mathbf{K}! \eta 2^{\mathbf{K}} \mathbf{K}! + o(1) \leq \frac{1}{2\mathbf{K}(\mathbf{K}-1)}$.

A Graph Theory result [5] states that a random k-uniform hypergraph with edges to variables ratio less than $\frac{1}{k(k-1)}$ is wpp a hyperforest. As functions in $\mathcal{V}$ are globally consistent, every hypertree has a solution. Therefore the instance we are left with at time t_ε is wpp satisfiable. The constraints remaining at time t_ε, do not form a **K**-regular hypergraph, as some constraints have arity less than **K**, but this only reduces the chances of cycles.□

Putting everything together we get that if $m < (\frac{1}{2\Gamma_{\mathcal{F}}} - \delta)n$, then there exists $\varepsilon > 0$ st wpp an instance of $BCSP_{n,m}(\mathcal{F})$ will be wpp reduced at time t_ε to an instance which is wpp satisfiable. And this proves Theorem 2.4.

2.6 Conclusions and Future Work

We introduced a slight variation of the random model for BCSP studied by Molloy. For a certain subclass of the model we proved a lower bound for unsatisfiability, which in some cases corresponds to a lower bound on the threshold for satisfiability.

The framework we used to prove the result is general enough to allow similar results for better algorithms. It is known that for SAT improved versions of Unit Clause algorithm, such as Shortest Clause [7] or Unit Clause with majority [2] have been used to prove tighter lower bounds. We believe a similar approach may be used for getting tighter lower bounds for our model.

The result may be possibly extended to CSP by modifying UC in such a way that it can deal with unit constraints over a non-boolean domain. The only feature which is required is to preserve the independence and uniform distribution of constraints in the classes of the process graph.

Bibliography

[1] D. Achlioptas. Setting 2 variables at a time yields a new lower bound for random 3-SAT. In *Proc. STOC-00*, pages 28–37, 2000.

[2] D. Achlioptas. Lower bounds for random 3-SAT via differential equations. *Theoretical Computer Science*, 265:159–185, 2001.

[3] D. Achlioptas, A. Chtcherba, G. Istrate, and C. Moore. The phase transition in 1-in-k SAT and NAE 3-SAT. In *Proc. SODA-01*, pages 721–722, 2001.

[4] D. Achlioptas and G.B. Sorkin. Optimal myopic algorithms for random 3-SAT. In *Proc. FOCS-00*, pages 590–600. 2000.

[5] B. Bollobás. *Random graphs.* Cambridge University Press, Cambridge, 2001.

[6] M. Chao and J. Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. Comput.*, 15(4):1106–1118, 1986.

[7] M. Chao and J. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k-satisfiability problem. *Inform. Sci.*, 51(3), 1990.

[8] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the *really* hard problems are. In *Proc. IJCAI-91*, pages 331–337, 1991.

[9] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statistics*, 23:493–507, 1952.

[10] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). In *Proc. FOCS-92*, pages 620–627, 1992.

[11] S.A. Cook. The complexity of theorem proving procedures. In *Proc. STOC-71*, pages 151–158. 1971.

[12] N. Creignou and H. Daudé. Generalized satisfiability problems: minimal elements and phase transitions. *Theoret. Comput. Sci.*, 302(1-3):417–430, 2003.

[13] N. Creignou and H. Daudé. Coarse and sharp transitions for random generalized satisfiability problems. In *Mathematics and computer science. III*, pages 507–516. 2004.

[14] N. Creignou, S. Khanna, and M. Sudan. Complexity classifications of Boolean constraint satisfaction problems. *SIGACT News*, 32, 2001.

[15] W. Fernandez de la Vega. On random 2-SAT, 1992. Manuscript.

[16] E. Friedgut. Sharp thresholds of graph properties, and the k-SAT problem. *J. Amer. Math. Soc.*, 12(4):1017–1054, 1999.

[17] A. Goerdt. A threshold for unsatisfiability. *J. Comput. System Sci.*, 53(3):469–486, 1996.

[18] G. Istrate. Coarse and sharp thresholds of boolean constraint satisfaction problems. 2005. To appear in Discrete Applied Mathematics.

[19] R. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. 1972.

[20] D.G. Mitchell, B. Selman, and H.J. Levesques. Hard and easy distributions of sat problems. In *Proc. AAAI-92*, 1992.

[21] M. Molloy. Models and thresholds for random constraint satisfaction problems. In *Proc. STOC-02*, pages 209–217, 2002.

[22] M. Molloy and H. Hatmani. Sharp threshold for constraint satisfaction problems adn homomorphisms, 2004. Manuscript.

[23] M. Molloy and M.R. Salavatipour. The resolution complexity of random constraint satisfaction problems. In *Proc. FOCS-03*, 2003.

[24] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. STOC-78*, pages 216–226, 1978.

[25] N.C. Wormald. Differential equations for random processes and random graphs. *Ann. Appl. Probab.*, 5:1217–1235, 1995.

Chapter 3

A variation of the random k-SAT model

3.1 Introduction

Satisfiability (SAT) is probably the most studied NP-complete problem. There are many different algorithms for solving it. If one wants to compare the performance of some solvers, one has mainly two alternatives for choosing the testbeds: use some of the "real-world" instances available at different web sites dedicated to SAT, or use randomly generated SAT instances. The advantage of the former is that one can test the solvers on instances more or less similar to the ones one would expect to encounter in reality. The disadvantage is that there are not too many such instances available. The lack of a "disciplined" approach to the study of structured instances is another important drawback.

Even though there are several different models for generating random SAT instances [27, 32, 29] the uniform random k-SAT model, introduced by Franco and Paull [11], is the one used in the large majority of cases. Its simplicity and amenability to analysis are among the most important reasons for this situation. As expected

there is a downside for these advantages: uniform random k-SAT lacks any kind of structure, which is usually associated with "real-world" instances. It has been noticed that solvers which perform very well on random k-SAT do poorly on "real-world" instances and vice-versa. The best example is Survey Propagation[25] which performs astonishingly well on random k-SAT and fails on almost all "real-world" instances [33].

In this chapter we propose a variation of the traditional random generator. As a new parameter is varied, instances are generated which interpolate between easy and exponentially hard subsets of NP-complete problems. The new parameter allows a finer tuning of the local dependency between variables, and thus may generate instances which are, in this respect, closer to structured instances. Still, it generates random instances, which preserves the simplicity and amenability to analysis of random k-SAT. These characteristics of the model allow us to get some theoretical results on the upper and lower bound for satisfiability. We have empirical evidence for a sharp phase transition in satisfiability, as the clauses to variable ratio increases. Thus the model has the main characteristics of random k-SAT. Still the handful of solvers we tested perform differently with respect to each other on instances generated by this model than they do on random k-SAT. Therefore we think this model generates valuable testbeds, as they may highlight solver characteristics which are not revealed by the random k-SAT instances.

3.2 General framework and definitions

Given a finite set of boolean variables and a boolean formula, the *Satisfiability problem (SAT)* asks to find a truth assignment of the variables which satisfies the formula. If no such assignment exists, the formula is proved to be *unsatisfiable*. A *literal* is a variable or the negation of a variable. We will sometimes refer to a literal

as a *signed variable.* The *sign* of a literal is 1, positive or TRUE if the literal is a variable, and 0, negative or FALSE if the literal is the negation of a variable. We will use $\neg$ to denote the negation operator. Therefore we will use $\neg v$ to denote the literal which represents the negation of v. A boolean formula is in *conjunctive normal form (CNF)* if it is a conjunction of disjunctions. Each disjunction is called *a clause.* The number of literals of a clause is referred to as *clause length.* We will use *k-clause* to refer to a clause of length k and $l_1 \vee \cdots \vee l_k$ to denote a k-clause on the set of literals $\{l_1, \ldots, l_k\}$. A *unit clause* is a clause consisting of only one literal. An *empty clause* is a clause of size 0; it can not be satisfied. A *pure literal* is a variable which appears only negatively or only positively in a formula. A *k-Not All Equal clause (k-NAE)* over the set of variables $\{x_1, \ldots, x_k\}$ is the conjunction of the clauses $x_1 \vee \cdots \vee x_n$ and $\neg x_1 \vee \cdots \vee \neg x_n$. We will use $\odot(x_1 \ldots x_k)$ to denote a k-NAE over $\{x_1, \ldots, x_k\}$.

Given $n, m, k, l, r \in \mathbb{N}$ such that $n/l \leq r \leq m$, $k \leq l \leq n$, the random *lk-SAT(m, n) model*[1] generates a random SAT instance with n variables and m k-clauses following the next steps:

(A) select uniformly at random r sets, called clusters, $R_1, \ldots, R_r$ of size l from the set of all n variables. The elements of a cluster are distinct, but the clusters may repeat.

(B) for each cluster R_i select uniformly at random η_i subsets, $S_1, \ldots, S_{\eta_i}$, of k elements, where[2] $\eta_i = \begin{cases} \lfloor m/r \rfloor, & \text{if } i \leq (m \bmod r\ ); \\ \lfloor m/r \rfloor + 1, & \text{else.} \end{cases}$

The elements within a subset are distinct, but the subsets may repeat.

(C) add a sign combination to each subset S_i, in such a way that clauses do not repeat within a cluster.

[1] When no confusion arises we will use *lk*-SAT to refer to random *lk*-SAT(m, n)

[2] We use m mod r to denote the remainder of the integer division of m by r.

Observation 3.1 We get a slightly different model if in rule (C) we do not ask for the clauses within clusters to be different. We will call this variation *independent lk-SAT*.

Observation 3.2 If $l = n$ lk-SAT is equivalent with random k-SAT.

3.3 Related work

3.3.1 NK-landscape

For $N, K \in \mathbb{N}$, $K \leq N-1$, an *NK-landscape* on N boolean variables $x_1, \ldots, x_N$ is a function $f : \{0,1\}^N \to \mathbb{R}$, $f(x_1, \ldots, x_N) = \sum_{i=1}^{N} f_i(x_i, V_i)$, where for each i $V_i \subset \{x_1, \ldots, x_N\}$, such that $|V_i| = K$, and $x_i \notin V_i$ and for each $i = 1, \ldots, N$ $f_i : \{0,1\}^{K+1} \to \mathbb{R}$. V_i is called the *the neighborhood of i*, and f_i is called *the fitness function of i*. We will consider only the *random neighborhood model*, in which, for each i the K neighbors are chosen at random. The *decision version of the NK-landscape problem* asks if the optimum of f equals N. It is NP-complete for $K \geq 2$[40, 39].

Culberson and Gao [14] introduced the following variation of a fitness function: given a parameter $z \in [0, 2^{K+1}]$ f^z is a binary function specified by randomly choosing z tuples from $\{0,1\}^{K+1}$ and define f_i to be 0 for the chosen tuples and 1 for all other tuples. They investigated the *fixed ratio binary NK-landscape*, which is defined as follows:

- it is a random neighborhood NK-landscape;
- it has an additional parameter $z \in [0, 2^{K+1}]$;
- if $z \in \mathbb{Z}$ for each $i \in 1, \ldots, N$, f_i is defined as f^z;

- if $z \notin \mathbb{Z}$, $z = \mu a + (1-\mu)(a+1)$, μn randomly chosen (without replacement) fitness functions will be defined as f^a and the rest will be defined as f^{a+1}, where $a = [z]$.

Culberson and Gao provided the following reduction from NK-landscape to $(K+1)$-SAT:

- the set of boolean variables is $\{x_1, \ldots, x_N\}$;
- for each f_i of type f^{z_i}, and each $j = 1, \ldots, z_i$ let C_i^j be a $K+1$ clause "such that for any assignment $y_j \in \{0,1\}^{K+1}$ that falsifies C_i^j, $f_i(y_j) = 0$"
- the SAT formula is $\bigwedge_{i=1}^{N} \bigwedge_{j=1}^{z_i} C_i^j$.

For $K = 2$, using the SAT reformulation they proved that for $z > 3$ the fixed ratio binary NK-landscape is unsatisfiable whp.

Observation 3.3 If $z = l/k$ and $K = k+1$ fixed ratio binary NK-landscape is very similar with lk-SAT. The only difference is that in lk-SAT the cluster R_i is chosen at random, while in NK-landscape the domain of f_i (which corresponds to the cluster) must contain x_i.

3.3.2 SAT instances

SAT instances are classified into *uniform random instances* and *structured instances*. There is no widely accepted definition of structure. As noticed in[18], "the notion of structure is very hard to define even though we recognize structure when we see it". In the absence of a good definition for structure, researchers have called *structured* any instance having a property which makes it statistically different than the uniform randomly generated instances. Ironically many structured instances are uniform random instances of other problems encoded to SAT. In these cases, the structure is "inherited" from the encoding schemes [20, 17].

Uniform random instances

The fixed-clause length random model is the most used one for generating random instances of SAT. This model is so common that the instances it generates are often called "random SAT" or "random formulas", even though there are other methods which also generate "random formulas" e.g. the random clause length model.

In the fixed clause length model m clauses of size k are created. Each clause is produced by selecting k distinct variables out of all n variables, and then negating each chosen variable with probability 1/2. The phase transition phenomenon has been noticed[26] for this model: as the clauses to variable ratio (CVR) is increased the probability of an instance to be satisfiable drops sharply from 1 to 0. An easy-hard-easy pattern of instance difficulty is associated with the phase transition. The hardest instances seem to appear in a very narrow region around the 50% solvability point (threshold). The threshold conjecture states that for each k, there is some r_k such that as n tends to infinity, any random k-SAT is satisfiable with probability tending to 1 when its CVR is less than r_k, and is unsatisfiable with probability tending to 1 when the CVR is bigger than r_k.

Closely related to the phase transition is the work on random k-SAT for resolution complexity. In their seminal paper[7] Chvátal and Szemerédi proved that for any $c > 0.7 \times 2^k$, whp any random k-SAT instance having n variables and cn clauses needs at least $(1+\epsilon)^n$ steps for the resolution to prove it unsatisfiable. Since then the result has been improved several times [12], and the complexity has been estimated more accurately. If an unsatisfiable formula has an exponential resolution complexity than any DPLL-like algorithm will need exponential running time to prove it unsatisfiable.

In an attempt to "challenge SAT solvers with modifications to Random 3-SAT to make instances more difficult", Bayardo and Schrag[2], studied two regular models: *variable-regular random 3-SAT* and *literal regular random 3-SAT*. In the former

case, m clauses are created by randomly selecting, without replacement, 3 distinct variables from a multiset containing $\lfloor 3(m/n) \rfloor$ copies of each of the n variables and a few unique randomly selected variables[3]. Each variable is negated with probability 1/2. In the latter case, the clauses are created by randomly selecting 3 variable-distinct literals from a set consisting of $\lfloor 3m/2n \rfloor$ occurrences of each literal, and some unique random literals.

It was noticed that both models exhibit a phase transition, but the threshold value is smaller (larger) than that of random 3-SAT for literal regular (variable regular) model. For both models the instances produced at their corresponding threshold are harder than the threshold instances of random 3-SAT, with literal regular random 3-SAT producing the hardest instances.

Observation 3.4 The original models of [2] were proposed for 3-SAT, but they can be naturally extended to k-SAT.

Observation 3.5 By having several clauses defined on the same cluster, lk-SAT is expected to produce instances which are "less regular" than the random k-SAT ones. Therefore lk-SAT can be viewed as opposite to the regular variable model.

Structured instances

There are two categories of structured instances *"real-world"* and *randomly generated structured instances.* "Real-world" instances come from problems for which SAT is the most natural representation, such as formal model verification, [35, 4] equivalence checking [6] and logic synthesis [5]. Such instances tend to be very big, in terms of variables, clauses and literals. There are very few such instances.

Many properties have been used as witnesses of structure, and thus used to randomly generate structured instances: the existence of well defined subproblems[27,

[3]With the goal of having exactly $3m$ variables in the set.

29, 32]; patterns of correlation among problem variables [16]; certain distributions of clause length and/or literals (usually resulting from converting instances of other problems to SAT [20, 17]).

There were two driving forces behind the inquiries to randomly generate structured instances: modeling instances which are somehow similar to the "real-world" ones and interpolating between different subsets of instances to study the hardness of the intermediate instances. (k, m)-trees[29] , chains[29], clustering[32], grid[16] and Quasigroup Completion [17, 1] are some examples falling in the first category, while 2+p[27], small world[38] and morphing[15] are the most known models in the second category.

In an attempt to generate more structured problems Rish and Dechter[29] proposed two new models for SAT instances: (k, m)-*trees* and *chains*. The former generates a tree of $(k+m)$-cliques such that any two neighbor cliques overlap on exactly k vertices. The later generates several uniform random k-SAT clusters (called subtheories) which are then connected in a chain by joining neighbor clusters with a 2-clause. Rish and Dechter noticed that the chain model generates a few instances which are very hard for DPLL based algorithms. Slater [32] suggested that the hardness of the chain instances is linked to "the combination of heuristics used and scale of the problems".

Along the same line as the two models above lays Slater's[32] *clustering method* for modeling more realistic instances; several disjoint subproblems are randomly generated and then they are linked together via a number of random clauses which have their literals chosen from the entire set of variables. Slater investigated the performance of several SAT solvers on instances generated by this model trying to estimate their "concentration - ability to concentrate on a subproblem". He concluded that heuristics derived from random k-SAT may not extend to other instances.

Observation 3.6 *lk*-SAT is similar to (k, m)-tree, chains and Slater's model as it creates random k-SAT subproblems on a subset of variables. It differs from them in the way it creates the links between these subproblems. In *lk*-SAT the links are "naturally" created by the random clusters, while in the above mentioned models the links are "artificially" created by adding clauses on top of the independent subproblems.

In[16] Ginsberg and McAllester proposed the *grid model* as a new method of generating structured instances. In this model, the variables of the instance are embedded on a square grid, and each clause is generated by choosing three neighbor variables which together form a right triangle on the grid. Instances generated according to this model are definitely more structured than the random ones, but it turned out that they are easily solved by most of the complete SAT solvers [2].

In the previous models random SAT instances are generated directly in such a way that some structure is created. A different approach is to generate uniform random instances of other problems and then convert them to SAT. In this case the structure of the SAT instance stems from the conversion and not from the original instance. Even though there are other problems used in such a scheme (Graph Coloring [20, 36, 15], Hamiltonian Cycle [20]) the Quasigroup Completion (QGC) is the best known example [17, 1].

QGC, introduced by Gomes and Selman [17], is the problem of completing a partially defined Latin square. A *Latin square* of size n is an $n \times n$ table, having each entry colored in one of n given colors in such a way that no color repeats in any row or column. The phase transition phenomena (as the fraction of filled cells is increased) and the associated easy-hard-easy pattern have been noticed [17, 1] for QGC. *Quasigroup with Holes (QGW)*[1] is a variation of QGC which generates only satisfiable instances. It still asks to fill a partial Latin square, which is generated by deleting some colors ("punching holes") in a Latin square. An easy-hard-easy

pattern (depending on the fraction of holes) has been noticed [1] for QWH. This is very surprising as the instances are guaranteed to be satisfiable. On the other hand this offers the possibility of varying the hardness of the satisfiable instances.

Watts and Strogatz[38] noticed that the connection graphs arising from "real-world" problems have a mixture of properties from random graphs and regular graphs; they have small average path length (as the random graphs), and high clustering (as the highly regular graphs). Based on this observation they proposed the *small world graphs*, as candidates for more structured "real-world" like graphs. Starting from a lattice graph, this model "gradually introduces disorder into the graph by rewiring each edge with probability p"[36]. Rewiring refers to replacing an edge by a new one which connects two randomly chosen vertices. Walsh [36] noticed that for Graph Coloring "small worldness" can have a large impact on search cost. Gent et al [15] noticed that many "real-world" SAT instances have the small world characteristic and proposed the model as a generator of "real-world like" instances.

The *"2+p"* model was proposed and investigated by Monasson et al[27] as a way of interpolating between 2-SAT, which is polynomial , and 3-SAT which is NP-complete. An instance with m clauses generated by this model has pm randomly generated 3-clauses and $(1-p)m$ randomly generated 2-clauses. Walsh extended this model to other problems: coloring, XOR-SAT, 1 in k SAT, NAE SAT [37].

Observation 3.7 If m/n is fixed and l is varied from k to n, lk-SAT interpolates between binary fixed ratio random NK-landscape and random k-SAT. While both problems are NP-complete, the first is typically polynomial[14] while the second can be (depending on n/m and the SAT solver) typically exponential[7]. In this respect lk-SAT is similar to $2+p$ as it interpolates between polynomial and exponentially hard instances. The difference is that in the case of lk-SAT the hardness refers to typical case, while for $2+p$ it refers to worst case.

Morphing is a general technique proposed by Gent et al[15] for generating instances which interpolates between a completely random one, and a highly structured one. They proposed 3 types of morphing, each of which essentially creates a new instance by repeatedly adding a new part to an existing structure. In the case of graphs, given two graphs of the same size and order, G_1 random and G_2 structured, a new one is created by selecting each edge from G_1 with probability p and from G_2 with probability $1-p$. Gent et al showed that $2+p$ and "small world" models can be viewed as particular cases of morphing.

3.3.3 SAT solvers

The research on SAT algorithms has been mainly focused on two classes of solvers: *complete methods* and *incomplete methods*. The former explore the entire search space and either come with a solution or prove there is no solution. The latter start with a complete assignment of variables and use local changes to move toward a solution. They either find a solution or report they could not find a solution. For this reason the incomplete methods are used only for satisfiable instances.

Complete solvers

Resolution [24] is the most used technique for obtaining proofs in boolean logic. Two clauses containing conflicting literals can not be satisfied by setting only the underlying variable of the conflicting literals. Therefore a new clause can be added consisting of all the literals of the two clauses other than the conflicting ones. The new clause will be called a *resolvant* of the two clauses. If a boolean formula has an empty resolvant than it is unsatisfiable. If all resolvants of a formula are non-empty than the formula is satisfiable. A boolean formula can have an exponential number of resolvants and therefore storing all of them is not possible.

DPLL[24, 23] avoids the exponential storage blow-up by using backtracking

search. It is presented in Fig. 3.1 following the description in[8]. Almost all complete SAT solvers are based on DPLL.

For any variable v and any sign σ, v_σ refers to the literal σv. For any literal l, $\phi(l)$ stands for the formula resulting by setting l to TRUE, discarding all clauses which contain l and deleting $\neg l$ from all clauses. Whenever a pure literal exists it is preferable to set the underlying variable accordingly, as this operation simplifies the formula without affecting its satisfiability. A unit clause needs to be satisfied immediately as it is the only possible way of finding a solution.

The combination of variable selection rule, in line 6, and sign (value) selection rule, in line 7, identifies the different complete SAT solvers. Here we will briefly describe three complete SAT solvers used in the experimental part of this chapter.

Satz[21] is a simple complete solver which performs consistently well on a large set of instances. It uses the *Unit Propagation (UP)* variable heuristic [28, 41], which chooses the next variable by assessing the combined effects of its instantiation and unit clause propagation. Excepting the forced literal (unit clauses) it does not use any value heuristic. Whenever a value is to be chosen it tries TRUE first and then FALSE.

Berkmin[10] is one of the best complete solvers for "real-world" SAT instances (its descendant ForkLift won the industrial category, at the 2003 SAT competition [3]). It was specifically designed and tuned to solve SAT instances obtained from circuit verification. It is known that it does not perform very well on random k-SAT instances [33]. It uses techniques such as conflict clauses learning and restarts (for details see [10]). A *conflict clause* is a clause deduced by resolution from the original clauses of an instance. The main variable selection rule chooses one of the variables appearing in the most recently added conflict clause. It also uses a second rule which chooses the variable having the largest number of occurrences in conflict clauses. Berkmin uses an intricate value selection rule which chooses either the value

```
Procedure DPLL (CNF formula φ)
1:if φ is empty return yes
2:elseif there is an empty clause in φ return no
3:elseif there is a pure literal u in φ return  DPLL(φ(u))
4:elseif there is a unit clause u in φ return  DPLL(φ(u))
5:else
6:      select a variable v from φ
7:      select a sign σ
8:      if DPLL(φ(v_σ) )=yes return yes
9:      else return DPLL( φ(¬v_σ) )
end
```

Figure 3.1: "Generic" DPLL procedure

producing the most 2-clauses or the value with the highest number of appearances in conflict clauses.

Kcnfs[9] is a complete solver specially designed for random k-SAT for which it is the best complete solver (it is the winner of the 2003 and 2004 SAT competitions[3] for the randomly generated instances). It is designed and tuned to be more effective on unsatisfiable instances. Its variable selection rule chooses the variable "for which both associated literals are strongly correlated with the others". The level of correlation of a literal l with the others is expressed in terms of the number of literals which have to be set to FALSE to force l to become TRUE. The actual formula is more complicated as a renormalization is needed to account for the different sizes of the clauses (for more details see [9]). The authors of Kcnfs noticed that a big number of binary clauses may have adverse effects on the solver. According to [9] they empirically tuned some parameters to avoid this situation. Kcnfs does not use any value ordering heuristic: it always tries the value FALSE first.

```
Procedure IR (CNF formula φ)
1:Guess a truth assignment T
2:while T does not satisfy φ
3:      if the iteration limit is exceeded return no
4:      Select a candidate set C of variables of φ
5:      Select a literal l from C
6:      Modify the truth assignment of l (and modify T accordingly)
7:end while
8:return yes
end
```

Figure 3.2: "Generic" stochastic SAT solver

Incomplete solvers

Incomplete methods can not prove unsatisfiability, but can be very useful for finding a solution. The general form an incomplete solver is presented in Fig. 3.2 following an example from[8]. Incomplete solvers start with a random assignment of variables. At each step, a set of candidate variables is chosen according to the candidates selection heuristic, in line 4. A variable selection heuristic, in line 5, is then applied to select the variable of the candidate set, which will be flipped (its value will be switched from TRUE to FALSE, or vice versa). If the new assignment satisfies the formula the algorithm reports the solution, otherwise it will repeat the same step until a predefined number of flips is exceeded. We will introduce three incomplete SAT solvers which will be used in the experimental part of this chapter.

Novel is a very efficient incomplete SAT solver which combines random walk [31] with a very carefully tuned literal selection heuristic: the Novelty heuristic[22]. It uses a very simple candidate selection heuristic: C contains the literals of an unsatisfied clause chosen at random. The variable selection heuristic considers all

the literals in the chosen clause, and for each one of them it assesses the number of unsatisfied clauses obtained by flipping it. The literals which produced the least unsatisfied clauses are taken in the best set. If no clause became unsatisfied by flipping, then l is chosen from the best set according to the tie-break heuristic. Otherwise, with probability p_w l is chosen at random from the entire set of literals of C, and with probability $1 - p_w$ it is chosen from the best set according to the tie-break heuristic. If the best set consists of a single literal the tie-break heuristic will choose it as l, unless it is the last chosen literal in which case, with probability p_s it will choose the second best move. If the best set contains several literals the tie-break heuristic will choose the least recently flipped literal.

ESGI[30] is a randomized solver for boolean linear programming, based on subgradient optimization[30]. It defines weights on clauses depending on their unsatisfiability. A penalty function is defined on a formula based on these weights. ESGI performs two "orthogonal" steps: primary steps, attempting at minimizing the penalty function, and dual steps, aiming at modifying the weights of the clauses. ESGI uses no candidate selection rule - the candidate set contains all variables. The literal selection rule chooses a candidate which, when flipped, reduces the value of the penalty function (primary step). Ties are randomly broken. If no such candidate exists (dual step), then the weights are modified using an exponential scheme and smoothed (for more details see [30]).

SPE is an implementation of the Survey Propagation algorithm[25], which performs astonishingly well on random instances up to the satisfiability threshold. Survey Propagation takes a different approach to solving SAT, based on results from statistical mechanics. The solution space is considered to be divided into several clusters. Within each cluster, some variables are fixed (have the same value in all solutions in the cluster) and other are unconstrained (in some solutions they are positive in other they are negative). The clauses are considered to send "preference"

messages to the variables they contain. A clause containing a positive literal will pass a preference for TRUE message to the underlying variable of that literal. The Survey Propagation algorithm tries to approximate the distribution of each variable over the values TRUE, FALSE and UNCONSTRAINED. It uses an iterative process starting with random values, hoping that the process will converge to the values corresponding to a cluster (SP phase). Even though there is no proof for the convergence of the process, it works very well in practice. If the process converges, a fixed number of the most biased variables (the ones taking values TRUE or FALSE with the largest probability), are set to the corresponding values (decimation phase). The instance is simplified accordingly and unit propagation is performed. If the unit propagation step proves the instance as unsatisfiable, the process is restarted from the beginning (there is a limit on the number of restarts). Otherwise a new round of SP and decimation phase is applied to the simplified formula. When there are no biased variables, the remaining formula is passed to ESGI, hoping that it will provide a solution.

3.4 Evolution for fixed l - theoretical Results

In this section we set l to a given value and consider the evolution of the model with m. The following Lemma describes the evolution of satisfiability with m.

Lemma 3.1 *For any n, r, l, k, s.t. $n \geq l \geq k$, if $m_1 > m_2$, then*

$$\mathbf{Pr}[x \text{ is } SAT\text{: } x \in lk\text{-}SAT(n, r, m_1)] \leq \mathbf{Pr}[x \text{ is } SAT\text{: } x \in lk\text{-}SAT(n, r, m_2)].$$

Proof: For any n, m, l, k, r

$$\mathbf{Pr}[x \text{ is SAT: } x \in lk\text{-SAT}(n, r, m_1)] = \sum_{\Theta \in \{0,1\}^n} \mathbf{Pr}[\text{x is satisfied by } \Theta]$$

Let $m_1 = m_2 + 1$, Θ be any truth assignment of the n variables, C_1 be an instance of lk-SAT(n, r, m_1), and C_2 be an instance of lk-SAT(n, r, m_2). There must exist $i \in \{1, \ldots, n\}$ s.t. $\eta_i(C_1) = \eta_i(C_2) + 1$ and, for all $j \in \{1, \ldots, n\} \setminus \{i\}$, $\eta_j(C_1) = \eta_j(C_2)$. As Θ is chosen uniformly at random, for any $j \neq i$

$$\mathbf{Pr}\,[\Theta \text{ satisfies } R_j(C_1)] = \mathbf{Pr}\,[\Theta \text{ satisfies } R_j(C_2)]$$

As $R_i(C_1)$ has more clauses than $R_i(C_2)$,

$$\mathbf{Pr}\,[\Theta \text{ satisfies } R_j(C_1)] \leq \mathbf{Pr}\,[\Theta \text{ satisfies } R_j(C_2)],$$

which implies the result for $m_1 = m_2 + 1$. The proof is then obtained by induction on the value of $m_1 - m_2$. □

The above lemma shows that as m increases the probability of solubility decreases. In the following subsections we show theoretical and empirical evidence for the next assertion: as m increases there is a phase transition in satisfiability from a region where all instances are satisfiable to a region where all instances are unsatisfiable with high probability (whp). The experiments indicate that this transition is sharp.

From this point on, we assume that r is equal to n.

3.4.1 A theoretical upper bound for satisfiability

In this subsection we prove the following:

Theorem 3.1 *For any $k \leq l \subset \mathbb{N}$, $\varepsilon > 0$, $m \geq (2^{k-l} - 1 + \varepsilon)n$*

$$\lim_{n \to \infty} \mathbf{Pr}\,[x \textit{ is SAT}\colon x \in lk\textit{-SAT}(n, r, m)] \to 0.$$

For the purposes of this subsection we will assume that clauses can repeat within clusters. This will make the clauses generated on the same cluster independent and thus simplify the analysis.

A proof similar to the one of Lemma 3.1 can be obtained for the following:

Lemma 3.2 *For any n, r, l, k, m, s.t. $n \geq l \geq k$*

$$\begin{aligned}&\mathbf{Pr}[x \text{ is SAT}: x \in lk\text{-SAT}(n, r, m)] \\ &\leq \mathbf{Pr}[x \text{ is SAT}: x \in \text{independent } lk\text{-SAT}(n, r, m)].\end{aligned}$$

Therefore any upper bound for satisfiability of independent lk-SAT is an upper bound for satisfiability of lk-SAT.

We call the set of clauses generated on the same cluster, *a package*. We say that a *package P freezes variable x ($P \succ x$)* if $P \Rightarrow x$ or $P \Rightarrow \neg x$. We say that *cluster C freezes variable x($C \succ x$)*, if its package freezes x. We say that *packages P_i and P_j conflict ($P_i \bowtie P_j$)*, if they freeze the same variable to contradictory values. We say that *clusters C_i and C_j conflict ($C_i \bowtie C_j$)* if their packages conflict.

Observation 3.8 If two clusters conflict with each other, then the instance is unsatisfiable.

Observation 3.9 By Lemma 3.1 it is enough to prove Theorem 3.1 for $m = (2^{k-1} - 1 + \varepsilon)n$, where ε is any positive constant. For this value of m there are $\lfloor n\varepsilon \rfloor$ clusters on which exactly 2^{k-1} clauses are generated. They are the only clusters we need to prove the result, and thus, we may assume that 2^{k-1} k-clauses are generated on each cluster we refer to.

Following the approach in [14] we will use the second moment method to prove that under this set up, whp there will be at least two clusters which conflict with each other. For this we will estimate the expected number of conflicts, and its variance.

We define

$$I_{ij} = \begin{cases} 1, & \text{if } C_i \bowtie C_j\ ; \\ 0, & \text{else.} \end{cases}$$

and $R = \sum_{1 \leq i,j \leq \varepsilon n} I_{ij}$.

The probability of getting a conflict from two clusters C_i, C_j of size l is:

$$\begin{aligned}\mathbf{Pr}[C_i \bowtie C_j] &= \sum_{h=1}^{l} \mathbf{Pr}[C_i \bowtie C_j : |C_i \cap C_j| = h] \times \mathbf{Pr}[|C_i \cap C_j| = h] \\ &= \sum_{h=1}^{l} \mathbf{Pr}[C_i \bowtie C_j : |C_i \cap C_j| = h] \times \frac{\binom{n}{l}\binom{l}{h}\binom{n-l}{l-h}}{\binom{n}{l}^2}\end{aligned}$$

By definition

$$\begin{aligned}&\mathbf{Pr}[C_i \bowtie C_j : |C_i \cap C_j| = h] \\ &= \mathbf{Pr}[(C_i \bowtie x_1) \textbf{ and } (C_j \bowtie x_2) \textbf{ and } (x_1 = \neg x_2) : |C_i \cap C_j| = h] \\ &= \frac{1}{2}\frac{h}{l^2}\mathbf{Pr}[C \succ x]^2\end{aligned}$$

Under the assumptions we made, $\mathbf{Pr}[C \succ x]$, does not depend on n, and henceforth we will denote it by $Pd(k,l)$. In Appendix A we present an approximation of $Pd(k,l)$.

It follows that

$$\begin{aligned}\mathbf{Pr}[C_i \bowtie C_j] &= \sum_{h=1}^{l} \frac{1}{2}\frac{h}{l^2} Pd(k,l)^2 \frac{\binom{n}{l}\binom{l}{h}\binom{n-l}{l-h}}{\binom{n}{l}^2} \\ &= Pd(k,l)^2 \frac{1}{2l^2\binom{n}{l}} \sum_{h=1}^{l} \left[h\binom{l}{h}\binom{n-l}{l-h} \right]\end{aligned} \tag{3.1}$$

We will now work on the last term of (3.1).

$$\sum_{h=1}^{l} h\binom{l}{h}\binom{n-l}{l-h} = \sum_{h=1}^{l} l\binom{l-1}{h-1}\binom{n-l}{l-h} = l\sum_{h=1}^{l}\binom{l-1}{h-1}\binom{n-l}{l-h}$$

Using a known combinatorial result [19] $\sum_{K=0}^{Q}\binom{Q}{K}\binom{S}{N-K} = \binom{Q+S}{N}$, and setting $Q = l-1$, $K = h-1$, $S = n-l$, $N = l-1$, we get

$$\sum_{h=1}^{l} h\binom{l}{h}\binom{n-l}{l-h} = l\binom{n-1}{l-1} = \frac{l^2}{n}\binom{n}{l} \tag{3.2}$$

Combining (3.1) and (3.2) we get:

$$\mathbf{Pr}[C_i \bowtie C_j] = Pd(k,l)^2 \frac{1}{2l^2\binom{n}{l}} \frac{l^2}{n}\binom{n}{l} = \frac{1}{2n} Pd(k,l)^2$$

Notice that $\mathbf{Pr}[I_{ij}=1] = \mathbf{E}(I_{ij}) = \mathbf{Pr}[C_i \bowtie C_j]$. It follows that:

$$\mathbf{E}(R) = \sum_{1\le i,j\le \varepsilon n} \mathbf{E}(I_{ij}) = \sum_{1\le i\ne j\le \varepsilon n} \mathbf{Pr}[I_{ij}=1] = \frac{(\varepsilon n)^2}{2n} Pd(k,l)^2 \qquad (3.3)$$

Thus, for any fixed l $\lim_{n\to\infty} \mathbf{E}(R) \to \infty$.

We want to use the second moment method, and to this end we want to prove that: $\lim_{n\to\infty} \frac{\mathbf{Var}[R]}{(\mathbf{E}(R))^2} \to 0$.

Since $R = \sum_{1\le i,j\le \varepsilon n} I_{ij}$, we have

$$\frac{\mathbf{Var}[R]}{(\mathbf{E}(R))^2} = \frac{\sum_{1\le i,j\le \varepsilon n} \mathbf{Var}[I_{ij}] + 2\sum_{(i,j)\ne(q,s)} (\mathbf{E}(I_{ij}I_{qs}) - \mathbf{E}(I_{ij})\mathbf{E}(I_{qs}))}{(\mathbf{E}(R))^2}.$$

As for any i, j such that $1 \le i,j \le \varepsilon n$, $I_{ij} \in \{0,1\}$,

$$\frac{\sum_{1\le i,j\le \varepsilon n} \mathbf{Var}[I_{ij}]}{\mathbf{E}(R)^2} \le \frac{\sum_{1\le i,j\le \varepsilon n} \mathbf{E}(I_{ij})}{\mathbf{E}(R)^2} = \frac{\mathbf{E}(R)}{\mathbf{E}(R)^2} = \frac{1}{\mathbf{E}(R)} \to 0.$$

We also have

$$\frac{\sum_{(i,j)\ne(q,s)} (\mathbf{E}(I_{ij}I_{qs}) - \mathbf{E}(I_{ij})\mathbf{E}(I_{qs}))}{\mathbf{E}(R)^2} = \frac{\sum_{(i,j)\cap(q,s)=\emptyset} (\mathbf{E}(I_{ij}I_{qs}) - \mathbf{E}(I_{ij})\mathbf{E}(I_{qs}))}{\mathbf{E}(R)^2} + \frac{\sum_{|(i,j)\cap(q,s)|=1} (\mathbf{E}(I_{ij}I_{qs}) - \mathbf{E}(I_{ij})\mathbf{E}(I_{qs}))}{\mathbf{E}(R)^2}.$$

If $(i,j)\cap(q,s) = \emptyset$, the two random variables I_{ij} and I_{qs} are independent. It follows that $\mathbf{E}(I_{ij}I_{qs}) - \mathbf{E}(I_{ij})\mathbf{E}(I_{qs}) = 0$

If $|(i,j)\cap(q,s)| = 1$ they have one element in common, say $j = q$. In this case, we have: $\mathbf{E}(I_{ij})\mathbf{E}(I_{qs}) = \mathbf{E}(I_{ij})^2$ and

$$\mathbf{E}(I_{ij}I_{qs}) = \mathbf{Pr}[I_{ij}I_{js}=1] = \mathbf{Pr}[I_{ij}=1]\mathbf{Pr}[I_{js}=1 : I_{ij}=1]$$

Given that C_i and C_j conflict with each other, the conditional probability that C_j and C_s conflict with each other can be easily computed by arguments similar to the ones used to compute the probability that C_i and C_j conflict. Thus we get: $\mathbf{Pr}[I_{js} = 1 \colon I_{ij} = 1] = \frac{1}{2n} Pd(k,l)$.

Putting everything together we get:

$$\begin{aligned}
&\frac{\sum_{(i,j)\neq(q,s)} (\mathbf{E}(I_{ij}I_{qs}) - \mathbf{E}(I_{ij})\mathbf{E}(I_{qs}))}{\mathbf{E}(R)^2} \\
&= \frac{\sum_{|(i,j)\cap(q,s)|=1} \frac{1}{4n^2} Pd(k,l)^3}{\left(\frac{(\varepsilon n)^2}{2n} Pd(k,l)^2\right)^2} - \frac{\sum_{|(i,j)\cap(q,s)|=1} \frac{1}{4n^2} Pd(k,l)^4}{\left(\frac{(\varepsilon n)^2}{2n} Pd(k,l)^2\right)^2} \\
&= \frac{\sum_{|(i,j)\cap(q,s)|=1} 1 - Pd(k,l)}{(\varepsilon n)^4 Pd(k,l)}
\end{aligned}$$

Since there are $\binom{\varepsilon n}{3}$ pairs (i,j), (j,s), it follows that:

$$\frac{\mathbf{Var}[R]}{\mathbf{E}(R)^2} = \frac{\binom{\varepsilon n}{3}}{(\varepsilon n)^4} \frac{1 - Pd(k,l)}{Pd(k,l)}$$

As $\lim_{n\to\infty} \frac{\binom{\varepsilon n}{3}}{(\varepsilon n)^4} \to 0$ we have that

$$\lim_{n\to\infty} \mathbf{Pr}[R = 0] \leq \lim_{n\to\infty} \frac{\mathbf{Var}[R]}{\mathbf{E}(R)^2} = 0.$$

Which proves the upper bound. □

3.4.2 A lower bound for unsatisfiability

We believe that for $m \leq 2^{k-2}n$ all instances are whp satisfiable. This would make $2^{k-2}n$ a lower bound for unsatisfiability.

If $m = 2^{k-2}n$ there will be 2^{k-2} clauses generated on each of the n clusters. In the worst case, the set of clauses generated on each cluster, will reduce to a 2-clause.

Thus the instance will have n random[4] 2-clauses, on n variables. Such an instance is exactly at the threshold of 2-SAT. The intuition is that if $m \leq 2^{k-2}n$ the instances will reduce to a SAT instance less constrained than a 2-SAT at the threshold, and thus it will be whp satisfiable.

We extensively tested this lower bound for different combinations of l and k and instances of up to 100,000 variables, and we didn't find any unsatisfiable instance. These results give empirical support to the previous lower bound assertion.

For the case $l = 3$, $k = 3$ we are able to prove that instances with $m = 2n$ are whp satisfiable.

A theoretic lower bound for unsatisfiability, $k = 3$, $l = 3$

In this subsection we will prove the following:

Theorem 3.2 *For $l = 3$, $k = 3$, $m = 2n$*

$$\lim_{n\to\infty} \mathbf{Pr}[x \text{ is } SAT\text{: } x \in lk\text{-}SAT(n, r, m)] \to 1.$$

Proof: Notice that if $m = 2n$, two different clauses will be generated on each cluster. By analyzing the possible sign combinations of the two clauses, we get that:

- with probability 3/7 the two clauses are equivalent with a 2-clause;
- with probability 1/7 the two clauses are equivalent with a 3-NAE clause;
- with probability 3/7 the two clauses are equivalent to a formula of the form $x \vee (\odot(y, z))$.

It follows that in this case, the lk-SAT model is equivalent with the $BCSP_{n,m}(\mathcal{F})$ model, where:

[4]The way the instances are generated makes it equally likely that any cluster reduces to any signed combination of two variables from that cluster

- $m = n$;
- $F = \{f_1, f_2, f_3\}$;
- f_1 is 2-SAT, $f_1^{-1}(0) = \{(0,0)\}$, $P(f_1) = 3/7$;
- f_2 is 3-NAE, $f_2^{-1}(0) = \{(0,0,0),(1,1,1)\}$, $P(f_2) = 1/7$;
- f_3 is defined by $f_3^{-1}(0) = \{(0,0,0),(0,1,1)\}$, $P(f_3) = 3/7$.

By Observation 2.12 we have that for any $\delta > 0$, $BCSP_{n,m}(\mathcal{F})$ is wpp satisfiable for $m < (\frac{1}{2T_{\mathcal{F}}} - \delta)n$, where $T_{\mathcal{F}}$ is the maximum over $[0,1]$ of the function:

$$h_{\mathcal{F}}(x) = \frac{1}{4}(1-x)\left(\frac{6x}{7} + \frac{18x}{7} + \frac{6}{7}\right)$$

Thus $h_{\mathcal{F}}(x) = \frac{3}{14}(1-x)(4x+1)$. $h_{\mathcal{F}}(x)$ is a binomial function which reaches its maximum for $x = 3/8$, and the maximum is $T_{\mathcal{F}} = 75/224$. Thus for any $\delta > 0$, if $m < (112/75 - \delta)n$ then $BCSP_{n,m}(\mathcal{F})$ is wpp satisfiable. As in our case $m = n$ it follows that the $BCSP_{n,m}(\mathcal{F})$ equivalent to lk-SAT is wpp satisfiable. By Observation 2.11 the equivalent $BCSP_{n,m}(\mathcal{F})$ has a sharp threshold, and therefore it is satisfiable whp, and so is lk-SAT. □

3.5 Evolution for fixed l - Empirical results

The main goals of this section are: to characterize the transition in satisfiability and difficulty of lk-SAT, and to compare the performance of some SAT solvers on the instances generated by this model.

In all the experiments presented in this section, we fixed k and l to certain values and let m/n vary in increments of 0.05. For each value of m we generated 100 test probes. We used three complete solvers: satz(SZ)[21], Berkmin(BM)[10] and kcnfs(KC)[9], and three stochastic search solvers: SPE [34], Novel[34] and ESGI[34,

k	l	empirical m/n at 50% satisfiable	satisfiability upperbound
3	3	2.67	3
3	4	3.42	3
3	5	3.73	3
4	4	6.85	7

Table 3.1: Empirical 50% satisfiability points

30]. The only common output that all these solvers report is the solving time (ST), which we used as a measure of the difficulty of the instances.

3.5.1 Empirical characterization of the model

For a few combinations of l and k, we empirically determined the 50% satisfiability point, and ran experiments for different values of n, letting m/n vary around the corresponding point(see Table 3.1 and Fig. 3.3).

For k=3, $l \geq 4$, the empirical values for the 50% satisfiability are larger than the upper bound for satisfiability. Rather than representing a contradiction, this points out the difference between asymptotic results and experimental results (on finite instances). The instances we experimented with are small, and thus they probably did not fail because of such a simple conflict as the one we studied.[5] The apparent contradiction is thus a result of not being able to perform "large enough" experiments. For the case k=4, l=4, the empirical value of the 50% satisfiability point we derived is within the bounds we assert, but the asymptotic value may be smaller.

All satisfiability plots have a cross-over point similar to the ones obtained for

[5] As presented in Appendix A, for $k, l \neq (3,3)$ the instances must have at least 46,000 variables for being trivially unsatisfiable at $m/n = 2^{k-1}$. Even for small values of l, such instances are out of reach for any existing solver.

problems which exhibit a sharp phase transition. Figure 3.3 presents the plots for different values of k and l. This seems to indicate that the lk-SAT model exhibits a sharp phase transition. For the case l=k=3, as pointed out in [13], the instances have a small (order $\log(n)$) 2-SAT subproblem which alone almost determines the satisfiability. As random 2-SAT has a sharp threshold we expect lk-SAT to show the same behavior. As l increases the instances become more and more similar to random k-SAT, and thus they will exhibit a sharp threshold as well. The type of analysis as in [14], does not carry over to the cases $k > 3$ or fixed $l > 3$, as it requires $r > n$. While this does not necessarily imply a coarse threshold, more theoretical analysis is needed to get insights into the kind of threshold these problems exhibit.

The cross-over point pictures we got are misleading. Theoretically the asymptotic threshold value should be the cross-over point. It seems that the model has several "local" cross-over points, which can be confused with the asymptotic one. This raises an interesting question: how many other empirical results, derived from "small" experiments suffer from the same drawback?

3.5.2 Comparison of solvers' performances

Even though the solvers (especially the randomized ones) have some adjustable parameters, we used the default values they came with.

For small values of l, the clauses generated on the same clusters are strongly correlated, and frequently they can be reduced to shorter clauses by resolution. For this reason we added a small preprocessing procedure, similar to the one used by satz. We called this procedure P. It performs limited resolution subject to the following restrictions:

- Resolvants have length at most k
- Unit clause propagation is performed

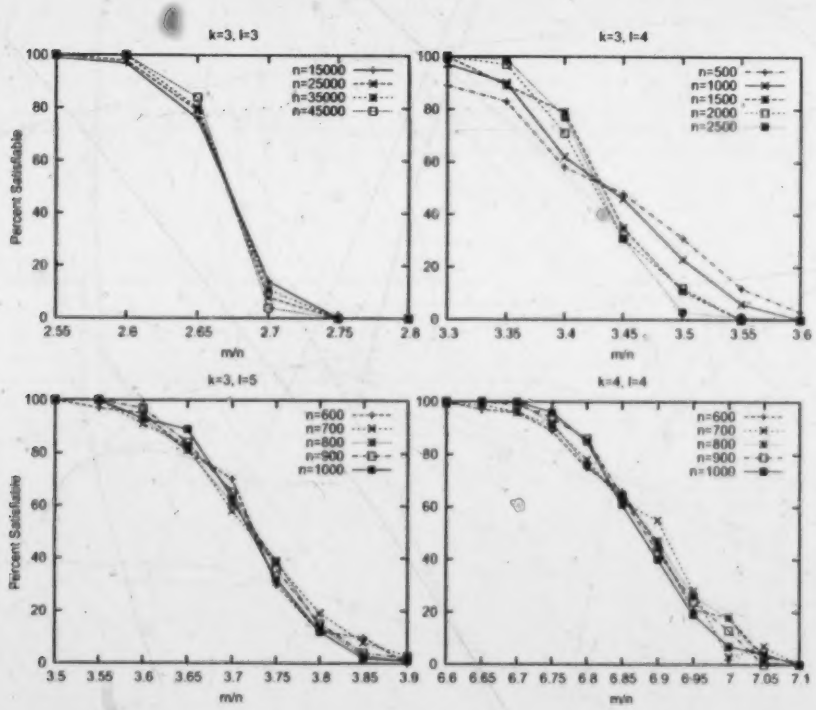

Figure 3.3: Evolution of satisfiability with m

- Resolvants will be no longer than any of the clauses they are derived from (the only exception is unit clause propagation)

- 2-clauses are only used to produce unit clauses

We noticed that P changes drastically the performance of some solvers so we also tested each algorithm combined with P[6]. For large values of n, P is done in half a second, while for smaller values it is done almost instantly, thus the time used by P does not contribute significantly to the total solving time.

All solvers were tested against the same sets of instances. Stochastic solvers were allowed to run for up to 200 seconds while complete solvers were allowed to

[6]For any solver we use P&solver to refer to the corresponding combination

k	l	n	m/n
3	3	5000,10000,15000,20000,25000	2.55,2.60,2.65,2.70,2.75,2.80,2.85
3	4	500,1000,1500,2000	3.30,3.35,3.40,3.45,3.50,3.55,3.60
3	5	500,600,700,800,900,100	3.55,3.60,3.65,3.70,3.75,3.80,3.85
4	4	500,600,700,800,900,100	6.70,6.75,6.80,6.85,6.90,6.95,7.00

Table 3.2: Values used in the experiments

Order	k=3,l=3	k=3,l=4	k=3,l=5	k=4,l=4
1	P&BM	P&KC,SZ	P&KC,SZ	P&SZ,SZ,P&KC
2	SZ,P&KC	P&BM,BM	KC,P&BM,BM	P&BM
3	BM	KC		BM

Table 3.3: Classification of complete solvers, according to solving time

run for up to 600 seconds. Nevertheless when we compare all the solvers on positive instances, we only report the instances solved by the complete solvers in less than 200 seconds.

Table 3.2 presents the values of n, m, k and l for which the experiments were performed. For each combination of values 100 instances were generated.

Complete solvers

Table 3.3 presents a summary of the experimental results for the complete solvers. We grouped together solvers which have comparable performances, and separate them if their solving times are significantly different (at least 3 times faster). Within a group we classify the solvers by the number of instances they solved. For the cases $l = k$, kcnfs was barely able to solve any instance even for the smallest values of n. For that reason we eliminated it from the corresponding columns in the tables.

Surprisingly, kcnfs the best solver for random k-SAT instances performs the worst of all the complete solvers we tested. On the other hand, P improved kcnfs quite a bit, making it the best or second best complete solver for each class. Our guess is that P reveals the characteristic of lk-SAT instances - random strongly interconnected components. As P produces shorter clauses on some clusters, it guides the search to those clusters, and then by unit propagation or a shallow search the solver is able to find a solution, or a quick proof of unsatisfiability. That is certainly the case for k=l=3, in which the 2-SAT subproblem almost determines the satisfiability. We think that for small values of l and k the 2-SAT subproblem and a (small) fraction of the remaining 3-clauses determine the satisfiability. As pointed out in Section 3.3.3 there is a problem with kcnfs when the formula has many 2-clauses. We don't understand kcnfs behavior, especially as P reveals the 2-clauses.

Even though Berkmin outperforms kcnfs in all cases, it still performs relatively poorly. P enhanced it especially in the case k=l=3, for which P&Berkmin is the best solver, outperforming all the others by almost an order of magnitude. We believe the 2-SAT subproblem is responsible for this behavior. Berkmin, which is specially designed for structured instances, makes good use of the 2-SAT subproblem revealed by P. This also explains why P&Berkmin is not so effective for the other cases: the 2-SAT subproblem is not so important and it still has to solve the remaining random 3-SAT component.

The performances of the complete solvers indicate that the cases k=3,l=4 and k=3,l=5 have similar characteristics. In both cases the solving time plots of all complete solvers (Fig. 3.4) have an easy-hard-easy shape, and the peaks in difficulty correspond to the 50% satisfiability points. The difficulty of the instances at the 'perceived' 50% satisfiability points, seems to be exponential in the number of variables (see Fig. 3.6). This is surprising and counter intuitive, as these points are beyond the asymptotic upper bound for satisfiability, and thus, asymptotically the

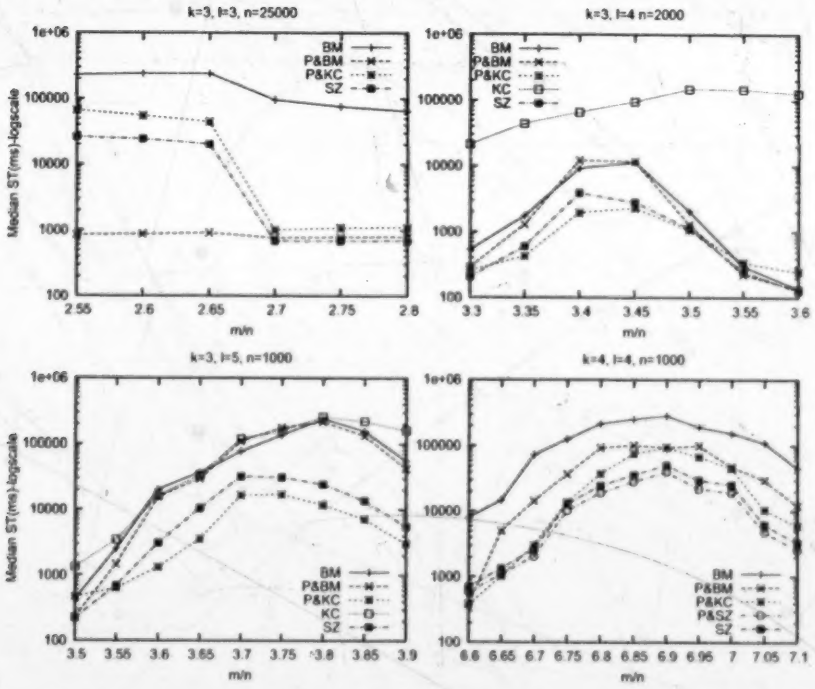

Figure 3.4: Evolution of difficulty with m

instances can be proved unsatisfiable in polynomial time.[7]

The cases k=l=3 and k=l=4, are not similar to each other. They are also different than the other 2 cases. For k=l=3, the difficulty of the instances decreases with increasing m. There is a sharp drop (for some solvers of an order of magnitude) in the difficulty, around the 50% satisfiability point (see Fig. 3.4). For all complete solvers the difficulty of the instances increases sub exponentially with n. For the instances below the threshold the difficulty seems to increase superlinearly with n (see Fig. 3.5). The plots look like polynomials of different degrees. Except for Berkmin, for which it is superlinear, the difficulty of the instances above the

[7] If the clusters are known it can be done in linear time. Otherwise all possible combinations of up to l variables can be checked, and there are $O(\binom{n}{l})$ such combinations.

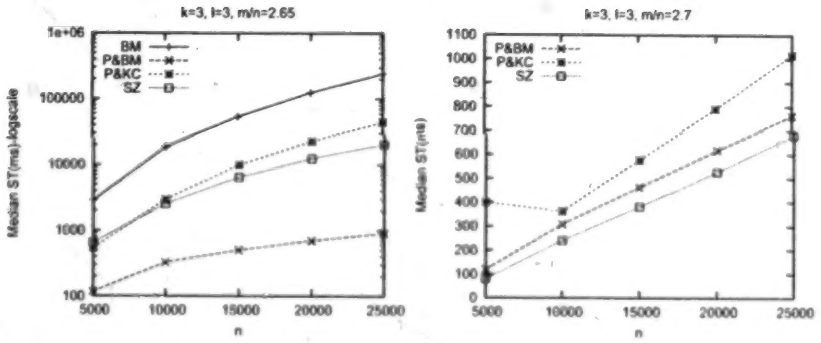

Figure 3.5: Evolution of difficulty with n, above and below threshold for $k = 3, l = 3$

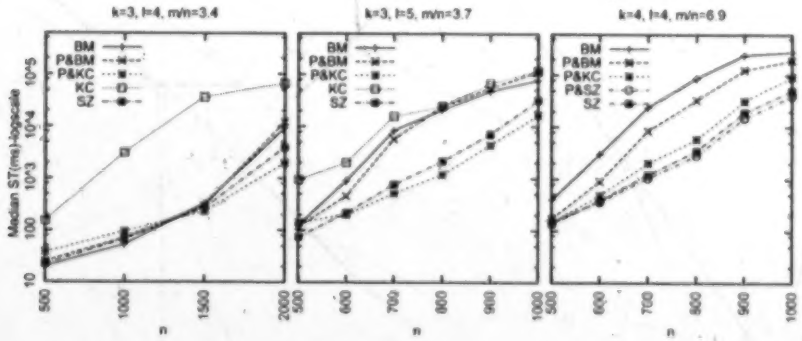

Figure 3.6: Evolution of difficulty with n, at the 50% satisfiability

threshold seems to increase linearly with n for all complete solvers. We believe the differences in solvers' performances is related to the amount of time they need to isolate the small 2-SAT subproblem. This also explains the efficiency of P in this case.

For the case k=l=4, the plots have an easy-hard-easy shape with the peak occurring at the 50% satisfiability point. The difficulty of the instances around the 50% satisfiability point seems to increase exponentially with n (Fig. 3.6). For this case we could not prove the existence of a small 2-SAT problem which almost determines

the satisfiability by itself. This leaves us with 2 possible scenarios:

- The instances have a 2-SAT subproblem which is not dense enough to determine the satisfiability. Thus longer clauses are needed to determine if the instance is unsatisfiable.

- The unit clauses are responsible for unsatisfiability (similar to the case which gave us the trivial upper bound).

The experiments indicate that the former is the case but, given the small size of the instances we experimented with, this may not be true asymptotically.

All solvers on satisfiable instances

The median solving times are not significantly different among the stochastic solvers on satisfiable instances. To get a better view of their performances we computed the percentage of satisfiable instances they were able to solve. Tables 3.4, 3.5, 3.6, 3.7 present these percentages for each solver.

Except for the case k=3,l=5, SPE performs very well despite the small size of some instances.[8] The good performance of SPE is very surprising given that it is based on the assumption of a locally 'tree like' connection graph. For small values of l it is certainly not the case, as the clauses generated on the same cluster are highly interconnected. Based on these experiments we think that SPE can be successfully used for solving instances which are not necessarily uniform random k-SAT.

Novel performs very well on small instances, but its performance decreases dramatically for large values of n. It is certainly true that the solving time of Novel increases superlinearly with n, and for some cases, k=3,l=4, it looks almost exponential. On judging the performance of Novel one must keep in mind that we used

[8]It is known that SPE suffers from boundary effects.

solver	n=5000	n=10000	n=15000	n=20000	n=25000
BM	100	100	100	100	1
P&BM	100	100	100	100	100
P&KC	100	100	100	100	100
SZ	100	100	100	100	100
ESGI	98	93	90	97	85
P&ESGI	99	97	95	95	94
SPE	99	99	96	97	96
P&SPE	99	95	87	87	79
Novel	84	76	66	56	51
P&Novel	96	93	90	90	88

Table 3.4: Percentage of satisfiable instances solved by each solver, for $k = 3$, $l = 3$

solver	n=500	n=1000	n=1500	n=2000
BM	100	100	100-	91
P&BM	100	100	100-	91
KC	100	100	79	44
P&KC	100	100	100	100
SZ	100	100	100	100
ESGI	100	94	85	71
P&ESGI	100	96	87	78
SPE	95	96	94	90
P&SPE	97	91	75	59
Novel	99	92	88	83
P&Novel	100-	95	90	82

Table 3.5: Percentage of satisfiable instances solved by each solver, for $k = 3$, $l = 4$

solver	n=500	n=600	n=700	n=800	n=900	n=1000
BM	100	100	99	91	74	62
P&BM	100	100	98	91	78	66
KC	100	100	100	99	92	79
P&KC	100	100	100	100	100	100
SZ	100	100	100	100	100	97
ESGI	100	100	95	91	89	87
P&ESGI	100	100	97	94	91	87
SPE	75	74	72	75	72	73
P&SPE	96	96	96	90	84	76
Novel	99	99	99	98	98	98
P&Novel	99	99	98	97	98	97

Table 3.6: Percentage of satisfiable instances solved by each solver, for $k = 3$, $l = 5$

solver	n=500	n=600	n=700	n=800	n=900	n=1000
BM	100	100-	95	80	58	42
P&BM	100	100-	98	87	73	61
P&KC	100	100	100	100	99	90
SZ	100	100	100	100	100	99
P&SZ	100	100	100	100	100	100
ESGI	100-	91	85	77	69	64
P&ESGI	100	92	88	78	66	64
SPE	100	100	100-	97	93	89
P&SPE	77	74	67	57	49	41
Novel	98	98	97	95	95	96
P&Novel	99	99	97	96	95	92

Table 3.7: Percentage of satisfiable instances solved by each solver, for $k = 4$, $l = 4$

the default values for the parameters which dictate the behavior of Novel+(probability of selecting second best move, and probability of random walk move).[9] It is known that Novel is very sensitive to the values of these parameters, but finding the optimal values seems to be a black art.

ESGI had the most surprising behavior: it performed very well on the k=l=3 instances, which are very large, and not so well on the instances from the other cases which were smaller. The instances for k=l=4 were the ones on which it failed most frequently.

For all combinations of l and k the best complete solvers outperformed the best stochastic solvers in terms of percentage of solved problems. For cases $k \neq l$ there were very few instances which have been solved by stochastic solvers but not solved by complete solvers, while the converse situation is more frequent.

For both complete and stochastic solvers, the performances varied across the different k, l pairs, and they are all different from the random k-SAT instances. We think this recommends the lk-SAT as an alternative testbed for SAT solvers, as it points out some features of the solvers, which the uniform random k-SAT model does not seem to catch.

3.5.3 Adding the preprocessing procedure

P improved drastically the performance of complete solvers. It improved Berkmin by 1 or even 2 orders of magnitude for the cases $l = k$. Kcnfs seems to benefit the most from P. For the cases k=l, kcnfs could barely solve any instances, while P&kcnfs was among the best 3 solvers in their classes. In the cases k=3,l=4 and k=3,l=5, P moved kcnfs from the last position to the first.

It seems that as l increases P becomes less effective for complete solvers. To

[9]We modified the maximum number of flips such that each variable gets a large number of chances to flip.

check this hypothesis we tested P&Berkmin and P&kcnfs on random 3-SAT. As can be seen in Figure 3.7, P enhances the performance of kcnfs on some instances and worsens it on others, but on average, its influence is negligible. Even though P can worsen the performance of Berkmin on some instances, on average it seems to improve a little bit the overall performance of Berkmin(see Fig. 3.8). Adding P allowed Berkmin to solve 3 more problems.

The stochastic solvers benefit from P, as well, especially for the large instances. Adding P allows them to solve on average 3% more problems. SPE is the only solver for which adding P worsens the results (except for the case k=3,l=5). The situation would have been even worse without restarts, as almost half of the problems solved by P&SPE needed several restarts, while SPE solved them in one try. Apparently the better SPE does the worse the performance of P&SPE. It also seems that the effect of P on SPE is correlated with the strength of the local dependency between variables. The stronger the dependency the worse the effect of P on SPE. To test these hypotheses we ran SPE and P&SPE on random k-SAT instances. Statistically, the effect of P on SPE for random instances is almost null. P&SPE solved 4 instances which SPE could not solve, while SPE solved one instance which P&SPE could not solve. Adding P to SPE changed the running time by at most $\pm$10%, which cannot be considered significant, especially as SPE is a randomized solver. It is also true that the effect of P on random k-SAT instances is not very important.

For the case k=3, l=5, P improves SPE, in terms of percentage of solved instances, even though it still worsens the running time. To make things even more complicated this is the only case when SPE made a good use of restarts. For all the other combinations of k and l, SPE solves an instance or gave up after 1 try (with very few exceptions). For k=3, l=5, SPE restarted several times, for almost all instances it could not solve. Nevertheless, if it found a solution, it did it in one try.

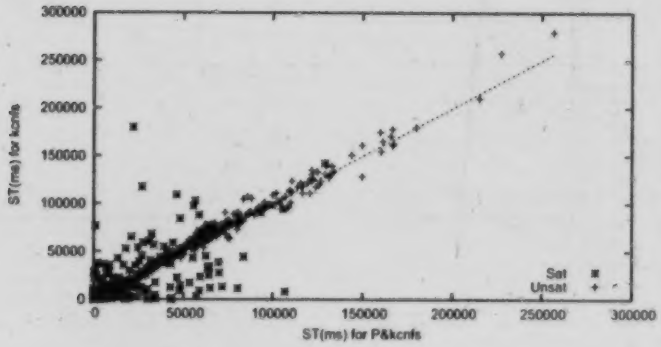

Figure 3.7: Scatter plot of ST for kcnfs versus ST for P&kcnfs for random 3-SAT

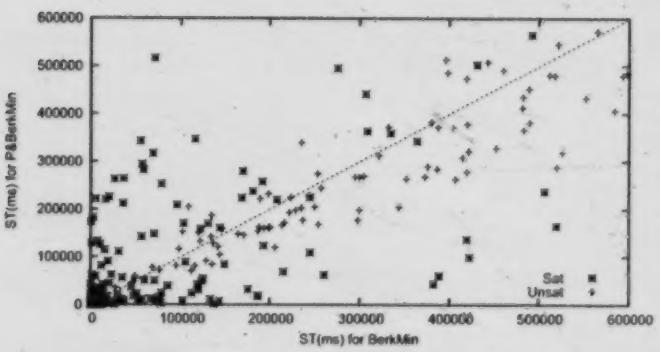

Figure 3.8: Scatter plot of ST for Berkmin versus ST for P&Berkmin for random 3-SAT

We guess that somehow P "reveals" the strong local variable dependency, which, by making the connection graphs less "tree-like", makes SPE fail. In spite of this, the overall conclusion is that P improves the performance of both complete solvers and stochastic solvers, for the lk-SAT instances without modifying it significantly for random instances. We think that P may be also useful for "real-world" instances, which usually have strong locally dependent variables.

3.6 Evolution with l

In this section we investigate the influence of l on the difficulty of lk-SAT instances. We reduce our attention to k=3, as it is the easiest case to analyze. For the purpose of this section, we consider m/n fixed to a certain value. We want to see how l affects the characteristics of lk-SAT instances under this setting. Intuitively for any fixed value of m/n, the probability that an lk-SAT instance is satisfiable should not decrease with l. As l increases the density of the local SAT component generated on the same cluster decreases, and thus it is more and more likely to be satisfiable. On the other hand, as l increases the clusters will overlap more and more, and thus the local components will "interact" more and more, making the instance less likely to be satisfiable. We believe that the local SAT components have a stronger influence on the satisfiability of the instance and thus as they become more likely to have a solution so does the instance they form. We present empirical evidence which supports our assertion.

In the previous section we noticed that regardless of the value of l, satz performed constantly well, so in this section we only used satz for testing purposes. We used both the solving time (ST) and the number of visited nodes (NVN)[10] to measure

[10]We only counted the nodes where satz made a decision, disregarding the ones for which the variables were forced to take a certain value.

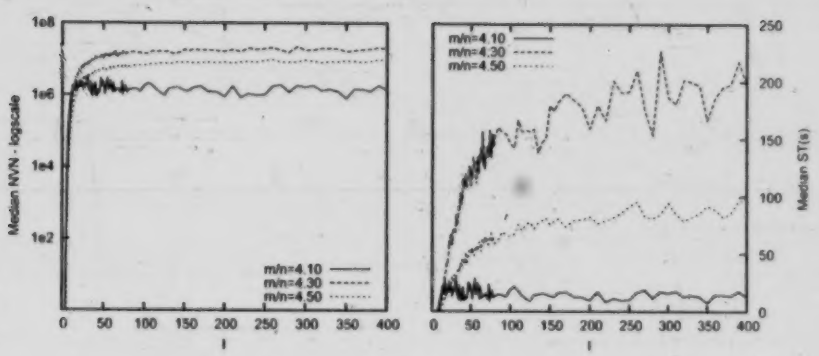

Figure 3.9: Evolution of hardness with l ($n = 400$)

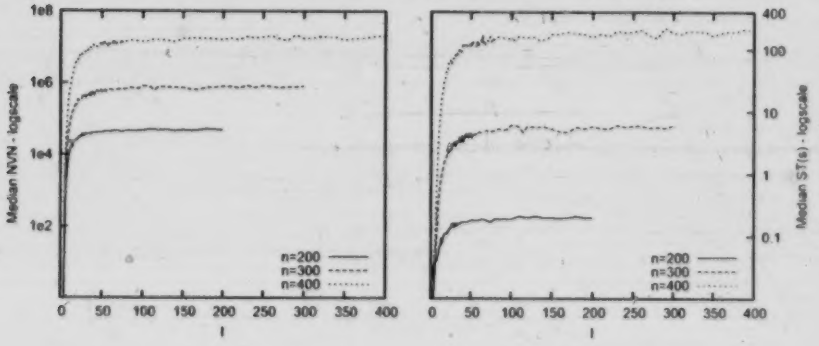

Figure 3.10: Evolution of hardness with l and n ($m/n = 4.30$)

the hardness of the instances.

In the first set of experiments, for fixed n and m, we let l vary from 3 to n. For each value of l we generate and solve 100 instances. The tests were performed for n=200, 300, and 400, and m chosen such that m/n varies from 3.8 to 4.8 in increments of 0.1.

For all values of n and m we plotted ST and NVN against l. Two well defined regions can be identified in all these graphics: for small values of l the instances are easy on average, resembling the binary fixed ratio NK-landscape; for large values of l the instances are very hard, similar to random 3-SAT. Within each of these regions the variation in the difficulty of the instances is small. In between these extremities there is a transition region, where the hardness of the instances increases steeply. We can see this pattern in both ST and NVN plots (see Fig. 3.10). The transition region seems to be narrow.

We noticed a slight difference between the plots of instances generated below and above the threshold of random 3-SAT (see Fig. 3.11). For the former the instances from the transition region are slightly harder than the random ones. For the latter the difficulty of instances from transition region increases gradually to the hardness of the random instances. We think this difference is a consequence of the stronger local dependency between variables which is associated with small values of l. Consider some instances having m/n <4.3. In the case of random 3-SAT, a DPLL-like search algorithm will quickly find a solution with almost no backtracking. In a similar lk-SAT instance for small l, the variables will be locally more clustered together. Therefore the search algorithm will encounter more inconsistencies and thus it will have to backtrack more often, spending more time on this instance. For the instances having m/n>4.3, the stronger local dependency between variables will help the algorithm discover the inconsistencies quicker, and thus it will visit less nodes.

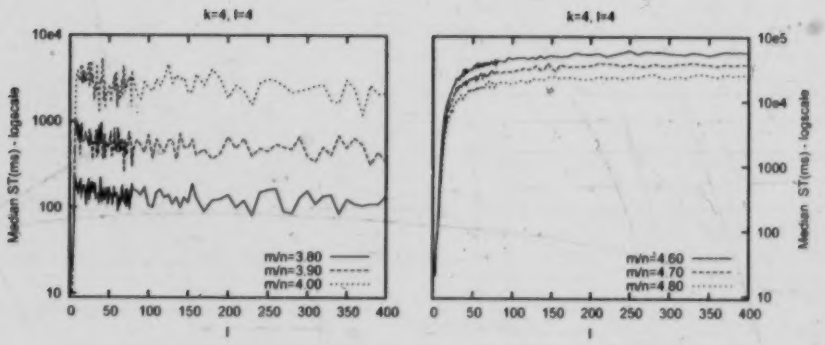

Figure 3.11: Different patterns in variation of hardness with l for $m/n < 4.3$ - left and $m/n > 4.3$ - right ($N = 400$)

3.7 Minimum unsatisfiable sets of lk-SAT instances

In this section we investigate if lk-SAT instances have small minimum unsatisfiable sets (MUS), which would make them easy to solve. We approximated MUS by the minimum of 3 minimal unsatisfiable subsets. The following procedure was used to determine a minimal unsatisfiable subset: at each step, a clause is randomly selected and eliminated from the instance; if this changes the satisfiability of the instance, the clause is added back. This step is repeated until all clauses have been selected. This should be a good estimation of the MUS, as the selection is biased toward small minimal unsatisfiable subsets. We chose to perform the experiments only for n=200. We recorded the number of clauses in the approximated MUS, which we called *the size of the MUS (SM)*. As can be seen in Figure 3.12 SM increases very rapidly with l and then stabilizes to values corresponding to the random instances. Thus we conclude that except for very small values of l, lk-SAT instances do not have a small MUS.

We noticed that SM, and ST (solving time) increase at different rates. In Fig. 3.12 two distinct regions can be identified; for small values of l SM increases rapidly

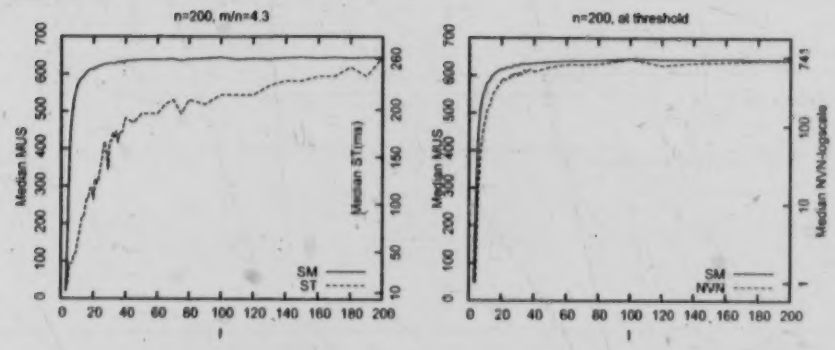

Figure 3.12: Different evolution of ST and SM (left), ST and NVN (right)

while ST is slowly increasing. As l increases, SM remains almost unchanged, while ST keeps increasing. We got a similar picture when we plotted log of NVN and MUS. One might find this very counter intuitive, especially if one considers the proof in [7], which links ST and MUS for uniform random k-SAT instances. For such instances, the size of the MUS and the NVN are highly correlated.

To further investigate the above results we performed a similar experiment for instances generated at the 50% satisfiability point. To get such instances the clusters were generated and then clauses were added to each cluster until the instance become unsatisfiable. For each instance we recorded the solving time and the approximated size of the MUS. We used the same method to approximate SM, but in these experiments we took 5 probes for each instance. Due to an initial error in our scripts, for a few values of l we ended up with 500 probes on the same instance. For those cases, SM was improved by less than 3%. This indicates that the approximation is trustworthy. The experiment was performed for n=200 and l between 3 and n. Again we saw the same difference in the growth of NVN and SM (see Fig. 3.12).

For small values of l ST is less than a second. As it is known that computers are not very reliable on timing very small time intervals, a possible cause of the difference in growth could be related to wrong timing. To check this hypotheses we

performed few similar experiments for n=400, and l between 0 and 20. We got the same picture even though ST was now in the safe range (more than one second for l larger than 6). Thus we conclude that the patterns of the growth of ST and SM are different for instances generated at 50% satisfiability point. We do not understand this behavior, but we guess that the stronger local dependency between variables reduces the complexity of proving a large MUS to be unsatisfiable.

3.8 Conclusions

We proposed a new model which preserves the main characteristics of uniform random k-SAT and produces instances on which some state of the art solvers perform differently than they do on random k-SAT instances. We presented empirical evidence that the model exhibits a sharp phase transition. We got theoretical upper and lower bounds for the satisfiability of the instances generated by this model. Besides their theoretical value, these bounds prove the amenability of the model to probabilistic analysis. We performed a thorough comparison of the performance of some SAT solvers on these instances and pointed out how it differs from their performance on random k-SAT and "real-world" instances. We think all these characteristics of the model recommend it as an alternative SAT testbed.

We also noticed a very interesting difference in the growth of the solving time and approximation of minimal unsatisfiable subproblem as l is varied.

We presented a preprocessing procedure which improved drastically the performance of almost all solvers we studied. More experiments need to be performed to see if this procedure is effective in the case of structured instances.

The lk-SAT model illustrates the danger presented by relying too much on random k-SAT for testing solvers performance: they will end up working very well on random k-SAT, but they will barely solve instances from a slightly changed distribu-

tion (as is was the case with kcnfs). The statement is reinforced by the efficiency of the preprocessing procedure: there is very little work needed in order to make solvers work, quite well, on *lk*-SAT. Still, without this work they perform very poorly.

Bibliography

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proc. AAAI-00*, pages 256–261, 2000.

[2] R.J. Bayardo and R. Schrag. Using CSP look-back techniques to solve exceptionally hard SAT instances. *Lecture Notes in Computer Science*, 1118:46–60, 1996.

[3] D. Le Berre and L. Simon. The essentials of the SAT 2003 competition. In *Proc. SAT'03*, pages 452–467, 2003.

[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS-99*, pages 193–207, 1999.

[5] R. K. Brayton. *Logic minimization algorithms for VSLI synthesis.* Kluwer, 1984.

[6] J. R. Burch and V. Singhal. Tight integration of combinational verification methods. In *Proc. ICCAD-98*, pages 570–576, 1998.

[7] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. Assoc. Comput. Mach.*, (4):759–768, 1988.

[8] S.A. Cook and D.G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In *Satisfiability Problem: Theory and Applications*, volume 35

of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. American Mathematical Society, 1997.

[9] G. Dequen and O. Dubois. kcnfs: An efficient solver for random k-SAT formulae. In *Proc. SAT-03*, pages 486–501, 2003. Kcnfs is available at http://www.laria.u-picardie.fr/~dequen/sat/.

[10] E.Goldberg and Y.Novikov. Berkmin: A fast and robust SAT-solver. In *Proc. DATE-02*, pages 142–149, 2002. BerMin is available at http://eigold.tripod.com/BerkMin.html.

[11] J. Franco and M. Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Appl. Math.*, 5(1):77–87, 1983.

[12] X. Fu. *On the complexity of prof systems.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1995.

[13] Y. Gao. Threshold phenomena in NK landscapes. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, 2001.

[14] Y. Gao and J. Culberson. An analysis of phase transition in NK landscapes. *Journal of Artificial Intelligence Research*, pages 309–332, 2002.

[15] I.P. Gent, H.H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proc. AAAI-99*, pages 654–660, 1999.

[16] M. L. Ginsberg and D. A. McAllester. GSAT and dynamic backtracking. *Lecture Notes in Computer Science*, 874:243–253, 1994.

[17] C. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proc. AAAI-97*, pages 221–226, 1997.

[18] C.P. Gomes. Structure, duality, and randomization: Common themes in AI and OR. In *Proc. AAAI-00*, pages 1152–1158, 2000.

[19] R. Graham, D.E. Knuth, and O.Patashnik. *Concrete mathematics*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[20] H.H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. IJCAI-99*, pages 296–303, 1999.

[21] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. IJCAI-97*, pages 366–371, 1997. Satz is available at http://www.laria.u-picardie.fr/~cli/satz214.2.c.

[22] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. AAAI-97*, pages 321–326, 1997.

[23] M.Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, (5):394–397, 1962.

[24] M.Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, (7):201–215, 1960.

[25] M. Mezard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812–815, 2002.

[26] D.G. Mitchell, B. Selman, and H.J. Levesques. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, pages 440–446, 1992.

[27] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. $2+p$-SAT: relation of typical-case complexity to the nature of the phase transition. *Random Structures Algorithms*, 15(3-4):414–435, 1999.

[28] P. W. Purdom, C. A. Brown, and E. L. Robertson. Backtracking with multi-level dynamic search rearrangement. *Acta Informatica*, 15(2):99–113, 1981.

[29] I. Rish and R. Dechter. Resolution versus search: two strategies for SAT. *J. Automat. Reason.*, 24(1-2):225–275, 2000.

[30] D. Schuurmans, F. Southey, and R. C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. IJCAI-01*, pages 334–341, 2001.

[31] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. AAAI-94*, pages 337–343, 1994.

[32] A. Slater. Modelling more realistic SAT problems. In *Proc. AJCAI-02*, pages 591–602, 2002.

[33] F. Southey. *Augmenting local search for Satisfiability.* PhD thesis, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004.

[34] F. Southey. Hybsat - a collection of SAT solvers, 2004. Available at http://www.cs.ualberta.ca/~finnegan/software.

[35] M. N. Velev and R. E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. In *Proc. DAC-01*, pages 226–231, 2001.

[36] T. Walsh. Search in a small world. In *Proc. IJCAI-99*, pages 1172–1177, 1999.

[37] T. Walsh. From P to NP: COL, XOR, NAE, 1-in-k, and Horn SAT. In *Proc. AAAI-02*, pages 695–701, 2002.

[38] D.J. Watts and S.H. Strogatz. Colective dynamics of 'small world' networks. *Nature*, (393):440–442, 1998.

[39] E. D. Weinberger. NP completeness of Kauffman's N-K model, a tunable rugged fitness landscape. Technical Report Working Papers 96-02-003, Santa Fe Institute, Santa Fe, 1996.

[40] A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of N-K fitness functions. Technical report, Department of Computer Science, University of Montana, 1999.

[41] R. Zabih and D. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *Proc. AAAI-88*, pages 155–160, 1988.

Chapter 4

On Comparing CNF SAT instances

4.1 Introduction

The models for generating SAT instances can be divided into two main classes: random and structured. Most of the structured models are uniformly distributed random instances of other NP-complete problems converted to SAT, e.g. Coloring [8, 13, 7], Constraint Satisfaction Programming[8], Hamiltonian Cycle[8], Subgraph Isomorphism. There are a few "standard" encoding schemes used to convert other problems to SAT, and thus the resulting instances have some common structural properties e.g. direct encoding produces long positive clauses and negative 2-clauses; log encoding tends to generate uniform length clauses.

The hardness of a model is expressed by the typical difficulty over well known solvers of its hardest instances, generated by choosing parameter values from a certain set or region known to generate very hard instances; usually around the corresponding threshold. The relative hardness of a model is **usually** obtained by comparing instances which have the same number of variables.

In this chapter we investigate the equal input length (number of literals) criterion for comparing CNF SAT instances. Using this criterion we empirically found the following conclusions which contradict common beliefs based on the same number of variables criterion:

(1) random 3-SAT is the hardest of the random k-SAT;

(2) log encoding of random Subgraph Isomorphism(SGI) produces instances which are harder than the corresponding random k-SAT;

(3) for random SGI, log encoding does not generate harder instances than direct encoding.

Even though the last two conclusions refer to SAT instances converted from SGI we believe they may apply to other structured instances. We conclude that for SGI, and it may be the case with many other problems, using only the same number of variables criterion to asses the relative hardness of instances is suspect and even misleading.

4.2 Motivation

We noticed that random Subgraph Isomorphism(SGI) instances converted to SAT are very hard (in terms of running time) for state of the art SAT solvers. To check this hypothesis we submitted to the 2003 SAT competition[4, 3] a set of several SAT instances obtained by converting [1] random SGI to SAT. The original SGI instances were solved by an SGI algorithm (Ullmann's [12]) in less then 10 minutes. None of the solvers competing in 2003 could solve the hardest SAT-translated SGI instances we submitted. Therefore, those instances were used at the 2004 competition [3], at

[1] The set contained instances obtained by applying both the direct encoding and log encoding (see Subsection 4.3.2).

which a few of them were solved by some stochastic solvers. None of the competing complete solvers could solve any SAT encoded SGI instance at the 2004 SAT competition.

In [5] seven of these instances were used to compare the efficiency of some state of the art SAT solvers. The instances have between 450 and 900 variables and between 28,000 and 71,000 clauses, almost all of them of size 2. The average running time of the best solver was 73 minutes. Therefore we can say that SAT solvers need a long running time to solve them. Despite this conclusion there are a few questions remaining: Can we say that they are hard SAT instances? What should we compare them against to get an answer? What comparison criteria should we use?

In comparing SAT instances (not necessarily their hardness) researchers typically rely on the same number of variables criterion [10, 7]. This criterion may be misleading for comparing SAT instances hardness. Based on this criterion it has been established that random 3-SAT is the easiest of random k-SAT. The meaning of the previous statement is that a random 3-SAT formula generated around its threshold is typically easier than a random k-SAT ($k \geq 4$) one generated around its own threshold and having the same number of variables as the random 3-SAT formula. There are strong theoretical and empirical arguments for this assertion.

According to this criterion we compare the typical hardness of a random 3-SAT with 100 variables to the typical hardness of for example random 8-SAT with 100 variables; both generated around their own threshold. For the random 3-SAT the threshold is at a clauses to variables ratio (CVR) of 4.26. For the random 8-SAT it is at a clauses to variables ratio of about 180. Thus we compare a 3-SAT instance having 426 clauses with an 8-SAT instance having 18,000 clauses. Things get worse when we consider the length of the clauses: the 3-SAT instance has 1,278 literals while the 8-SAT one has 144,000 literals, more than a hundred times more. Thus it is no surprise that the random 8-SAT is harder!

Different encoding schemes of other problems to SAT are compared on the "same native instance" basis. Again we see a huge difference in the number of literals of the formulas which are compared.

Thus it appears natural to ask: What would be the result of these comparisons if we force the formulas to have the same input length?

In complexity theory the input length is used as the basis of analysis; in that case the input is considered to be binary encoded. In our case binary encoding will correspond to a reduction in size proportional to the log of the number of variables. As our inquiry has only empirical goals, and as almost all SAT solvers use 16-bit integers to represent literals, we will not consider binary encodings.

The influence of the size of an instance on its hardness has been noticed for stochastic solvers [8]. It is generally accepted that the shorter a hard instance, the harder it is; the SAT competition [3] awards as the hardest instance the unsolved one which has the smallest input length, not the one that has the fewest variables.

While better representations of SAT formulas may exist, CNF is the standard representation. From this point of view we can state that our inquiry applies to SAT formulas in general.

4.3 Definitions, background knowledge and related work

In this section we introduce the Subgraph Isomorphism Problem and a few results regarding it. As several definitions and techniques for Subgraph Isomorphism have been adapted from their Constraint Satisfaction problem counterparts, we start by introducing a few definitions and properties of the Binary Constraint Satisfaction problem. We also present DCS, a new SAT solver, which will be used in the experimental part of this chapter.

4.3.1 Binary Constraint Satisfaction Problem

In this subsection we follow the definitions and notations from [6] and[11]. A *constraint satisfaction problem (CSP)* consists of a finite set of variables, each associated with a domain of values, and a set of constraints. Each *constraint* is a relation, defined on a subset of the variables, called its *scope*, denoting their legal combination of values. A solution is an assignment of a value to each variable from its domain such that all the constraints are satisfied. Typical CSP's are to determine whether a solution exists, to find one or all solutions.

In a *uniform domain* CSP, all variables have the same domain. A *Binary CSP (BinCSP)* has only binary constraints. A *binary constraint* is *arc consistent* if and only if any assignment of one variable can be extended to a consistent assignment of the other variable in the constraint scope. A *BinCSP* is *arc consistent* if and only if any constraint is arc consistent.

Algorithms for solving CSP, are mainly backtracking algorithms which maintain some kind of consistency at each node in the search tree. *Maintaining arc consistency (MAC)* is a BinCSP solver which at each step enforces arc consistency. *Forward checking* is similar to MAC, but at each step it enforces arc consistency only on those constraints which contain the most recently instantiated variable and the ones which are still to be instantiated. One of the most used variable heuristics is *Fail First* which prefers the variable with the smallest remaining domain.

4.3.2 Subgraph Isomorphism

We consider finite, undirected, loopless graphs without multiple edges.

A *graph* G is defined as a pair $G = (V_G, E_G)$, with V_G being the set of *vertices* of G, and E_G being the set of *edges* of G. E_G is a set of unordered pairs of distinct elements of V_G. The number of vertices of a graph G is referred to as *the order of*

the graph G and we will use n_G to denote it. The number of edges of a graph G is referred to as *the size of the graph*, and we will use m_G to denote it. Two vertices v and w of a graph G are *adjacent* if they are connected by an edge, $(v, w) \in E_G$. If v and w are adjacent we say that v *is a neighbor of* w. If v and w are not adjacent we say that v *is a non-neighbor* of w. A *non-edge* of a graph is a pair of distinct vertices which are not adjacent. The *degree of a vertex* is the number of its neighbors: $deg_G(v) = |w \in V_G : (v, w) \in E_G|$. For $n \in \mathbb{N}^*$, $m \in \{0 \ldots \binom{n}{2}\}$, the G_{nm} *random graph model* generates graphs of order n, and size m. The edges are randomly chosen without return from the set of all possible edges.

A *subgraph* G *of a graph* $H = (V_H, E_H)$, is a graph (V_G, E_G) such that $V_G \subseteq V_H$, and $E_G \subseteq E_H$. Two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are *isomorphic*, if and only if there exists a bijective mapping $f : V_G \to V_H$ such that for each $v_1, v_2 \in V_G$, $v_1 \neq v_2$, $(v_1, v_2) \in E_G$ if and only if $(f(v_1), f(v_2)) \in E_H$.

A graph G is *isomorphic to a subgraph of a graph* H, if and only if, there exists a subgraph H_1 of H, such that G is isomorphic to H_1.

Given two graphs $G = (V_G, E_G)$, $H = (V_H, E_H)$ the *subgraph isomorphism problem (SGI)* asks if G is isomorphic to a subgraph of H. If the answer is yes, then a subgraph(H_1) of H to which G is isomorphic, as well as a bijective mapping of G to H_1 which defines the isomorphism, should be provided. Given an SGI instance asking if G is a subgraph of H, we will use "the graph" to refer to H and "the subgraph" to refer to G. We will use $\{w_1, \ldots, w_{n_H}\}$ to refer to the vertices of the graph, and $\{v_1, \ldots, v_{n_G}\}$ to refer to the vertices of the subgraph; both sets of vertices are considered ordered.

Observation 4.1 It is very straightforward to convert an SGI instance to a BinCSP. The CSP variables correspond to the vertices of the subgraph and the values correspond to the vertices of the graph. For each pair of BinCSP variables which correspond to adjacent vertices in the subgraph a binary constraint is added allowing

only those values which correspond to adjacent vertices in the graph.

For $n, m \in \mathbb{N}$, $p \in \{0 \ldots \binom{n}{2}\}$, $q \in \{0, \ldots, \binom{m}{2}\}$ a *random* (n, m, p, q) *SGI* consists of a graph H and a subgraph G, where H is a random G_{mq} graph and G is a random G_{np} graph. As random SGI has 4 parameters, it is difficult to analyze it when all parameters are allowed to vary. In most cases three parameters are fixed and only one is allowed to vary. Usually n, m and q are fixed and p is varied, from 1 to $\binom{n}{2}$; to get interesting instances n is chosen such that $n \leq m$. In this case, as p increases there is a phase transition in the solubility of SGI. The transition in solubility is very sharp. Experiments indicate that the instances hardness follows the easy-hard-easy pattern. For fixed m, the hardness of the threshold instances (as measured by the running time of SGI solvers) depends heavily on the values of n and p.

Most of the algorithms designed to solve SGI consist of two parts: *a refinement procedure* and *a backtracking style tree search.* The refinement procedure is intended to propagate the effect of the partial solution on the future choices. Most of the effort for solving the problem has been directed toward the refinement phase. The tree search is usually performed by a chronological backtracking, endowed with variable heuristic.

The most efficient general algorithm for random SGI is the one proposed in 1976 by Ullmann[12]. It uses a backtracking approach for exploring the search tree. The refinement procedure enforces arc consistency after each variable instantiation. The algorithm creates and maintains an $n_G \times n_H$ matrix M which keeps the possible mappings of vertices of G into vertices of H. A 0 in row i column j of matrix M, means that vertex v_i can not be mapped into vertex w_j, while a 1 in the same position means that the mapping is possible.

The algorithm starts with the matrix M defined by the following equation:

$$M_{ij} = \begin{cases} 1 \text{ if } deg_G(v_i) \leq deg_H(w_j) \\ 0 \text{ otherwise} \end{cases}$$

The matrix M is then made arc consistent. This means that any M_{ij} for which there is a vertex $v_l \in V_G$ such that $(v_i, v_l) \in E_G$ and for any vertex $w_k \in V_H$, $(w_j, w_k) \notin E_H$, is set to 0. This rule is based on the observation that any mapping of v_i to w_j which will force a neighbor of v_i to be assigned only to non-neighbors of w_j will not produce a solution, and thus it can be eliminated.

The refinement procedure that Ullmann proposed checks any possible mapping against this rule. The procedure is reiterated until no change is made in the matrix. If at any moment there is a row i in the matrix M, that consists only of 0's, then there is no solution for the current (partial) problem. Ullmann noticed that if the matrix M is bit-packed than the refinement procedure reduces to several "bit and" and "bit or" operations. Our experience with Ullmann's algorithm indicates that it "barely" handles the hardest instances with $m = 32$.

Observation 4.2 Considering the conversion presented in Observation 4.1, Ullmann's algorithm can be viewed as a very fast (due to the "bit logic" operations) implementation of MAC on the corresponding BinCSP.

There are two straightforward conversions of SGI to SAT: the direct encoding and the log encoding.

For each $v_i \in V_G$ and each $w_j \in V_H$ *the direct encoding* defines a boolean variable b_{ij}. The variable b_{ij} is assigned TRUE, if and only if v_i will be mapped into w_j. The direct encoding will produce a SAT instance with $n_G n_H$ variables.

For each $v_i \in V_G$, a clause of the form

$$b_{i1} \vee b_{i2} \cdots \vee b_{in_H}$$

is added, to make sure that each vertex of G is mapped into some vertices of H. There are n_G such clauses.

For each $v_i \in V_G$, and each pair $(j < k) \in \{1, \ldots, n_H\}^2$ a clause of the form

$$\neg b_{ij} \vee \neg b_{ik}$$

is added to insure that each vertex of G is mapped into at most one vertex of H. There are $n_G n_H(n_H - 1)/2$ such clauses.

$n_H n_G(n_G - 1)/2$ clauses are added to guarantee that at most one vertex of G is mapped into any vertex of H. The clauses are of the form:

$$\neg b_{ji} \vee \neg b_{ki},$$

where $i \in \{1, \ldots, n_H\}$, $(j < k) \in \{1, \ldots, n_G\}^2$.

For each pair $(i < j) \in \{1, \ldots, n_G\}^2$ such that $(v_i, v_j) \in E_G$ and each pair $(k < l) \in \{1, \ldots, n_H\}^2$ such that $(w_k, w_l) \notin E_H$ the following clauses are added:

$$\neg b_{ik} \vee \neg b_{jl}$$

$$\neg b_{il} \vee \neg b_{jk},$$

to make sure that no edge of the subgraph is mapped into a non-edge of the graph. There are $m_G(n_H(n_H - 1) - 2m_H)$ such clauses.

Therefore, the direct conversion of an SGI creates a SAT instance which has:

- $n_G n_H$ variables;
- n_G clauses of size n_H, containing only positive literals;
- $n_G n_H(n_G + n_H - 2)/2 + m_G(n_H(n_H - 1) - 2m_H)$ 2-clauses with negative literals.

Observation 4.3 The input size of the direct encoding of a SGI is:

$$n_G n_H(n_G + n_H - 1) + 2m_G(n_H(n_H - 1) - 2m_H).$$

To introduce the *log encoding*, we need the following definition: for any $j \in \mathbb{N}$, $l \in \mathbb{N}^*$, $i \in \mathbb{N}^*$, such that $j < 2^l$, $i \leq l$, and any boolean variable b we define $\mathbf{nsgn}_l^{i,j}(b)$ to be the variable b signed by the complement of the i-th bit of the binary representation of j on l bits.

$$\mathbf{nsgn}_l^{i,j}(b) = \begin{cases} \neg b \text{ if the } i\text{-th bit of } j \text{ representation on } l \text{ bits is 1} \\ b \text{ if the } i\text{-th bit of } j \text{ representation on } l \text{ bits is 0} \end{cases}$$

The log encoding will generate a SAT instance with $n_G l_H$ variables, where $l_H = \lceil \log_2 n_H \rceil$. For each $v_i \in V_G$, l_h boolean variables $b_{i1}, \ldots, b_{il_H}$ are added. Variable b_{ij} is TRUE if and only if v_i can be mapped into w_{k+1}, and the j-th bit of the binary representation of k is set to 1.[2]

If n_H is not a power of 2, then the values above $n_H - 1$ must be avoided. To this end, for each $v_i \in V_G$ and each $d \in \{n_H, \ldots, 2^{l_H} - 1\}$ a clause is added, having the form:

$$\mathbf{nsgn}_{l_H}^{1,d}(b_{i1}) \vee \mathbf{nsgn}_{l_H}^{2,d}(b_{i2}) \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,d}(b_{il_H}).$$

There are $2^{l_H} - n_H$ such clauses.

To make sure that each vertex of G is mapped into at most one vertex of H, for each pair $(i < j) \in \{1, \ldots, n_G\}^2$ and each $k \in \{0, \ldots, n_H - 1\}$ a clause of the form:

$$\mathbf{nsgn}_{l_H}^{1,k}(b_{i1}) \vee \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,k}(b_{il_H}) \vee \mathbf{nsgn}_{l_H}^{1,k}(b_{j1}) \vee \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,k}(b_{jl_H})$$

is added. There are $n_H n_G(n_G - 1)/2$ such clauses.

For each pair $(i < j) \in \{1, \ldots, n_G\}^2$ such that $(v_i, v_j) \in E_G$ and each pair $(k < l) \in \{0, \ldots, n_H - 1\}^2$ such that $(w_{k+1}, w_{l+1}) \notin E_H$ the following clauses are added:

$$\mathbf{nsgn}_{l_H}^{1,k}(b_{i1}) \vee \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,k}(b_{il_H}) \vee \mathbf{nsgn}_{l_H}^{1,l}(b_{j1}) \vee \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,l}(b_{jl_H})$$

$$\mathbf{nsgn}_{l_H}^{1,l}(b_{i1}) \vee \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,l}(b_{il_H}) \vee \mathbf{nsgn}_{l_H}^{1,k}(b_{j1}) \vee \cdots \vee \mathbf{nsgn}_{l_H}^{l_H,k}(b_{jl_H}),$$

[2]Because in the normal binary representation on l bits the numbers from 0 to $2^l - 1$ are represented, the indexes of V_H are shifted to the left by a unit.

to make sure that no edge of the subgraph is mapped into a non-edge of the graph. There are $m_G(n_H(n_H-1)-2m_H)$ such clauses.

Therefore the log encoding of a SGI is a SAT instance with:

- $l_H n_G$ variables;
- $2^{l_H} - n_H$ clauses of size l_H;
- $n_H n_G(n_G-1)/2 + m_G(n_H(n_H-1)-2m_H)$ clauses of size $2l_H$.

Observation 4.4 If n_H is a power of 2 then the log encoding will produce a SAT instance having only clauses of size $2l_H$.

Observation 4.5 The input size of a log encoding of a SGI is:

$$l_H(2^{l_H} - n_H + n_H n_G(n_G-1) + 2m_G(n_H(n_H-1)-2m_H)).$$

Observation 4.6 If $m < (\log_2 m - 1)(n-1)$ then the input size of the direct encoding of a random (n, m, p, q) SGI is smaller than the input size of the log encoding of the same instance. This inequality holds for almost all interesting random SGI.

Observation 4.7 Even though instances generated by the direct encoding have more variables than the ones generated by the log encoding, their input size may be (and usually is) smaller than the input size of the log encoded ones.

Walsh [14] investigated several back and forth conversions between SAT and BinCSP. Walsh showed that Unit Propagation is more effective for direct encoding than it is for log encoding, when the two encoding schemes are applied to the same original BinCSP.[3] In [14] it is also shown that DPLL applied to the direct encoding of a BinCSP is as efficient (in terms of visited nodes) as forward checking applied to

[3]Walsh's result holds for the general CSP.

the original BinCSP instance, under equivalent branching heuristics. Under similar assumptions, forward checking applied to BinCSP is more efficient than DPLL applied to the log encoding of the original BinCSP. Given the close relation between BinCSP and SGI we believe that similar results will be true for SGI.

For the same random SGI, the log encoding produces SAT instances which are harder than the ones generated by the direct encoding. Both encoding schemes produce very hard SAT instances, as measured by the SAT solvers' running time [3].

4.3.3 The DCS SAT solver

DCS is a new SAT solver recently introduced by Culberson et al[5]. While it uses the traditional approach of a backtracking style tree search, it uses a novel variable heuristic, based on a special representation of a SAT instance. DCS was inspired by a direct solver for the Dominating Clique Problem. It assumes that the SAT instance has only positive clauses of any size and negative 2-clauses[4]. It tries to satisfy the positive clauses by setting some of their literals TRUE, and then it uses the negative 2-clauses for Unit Propagation.

It uses a very simple variable heuristic: it chooses the variable from the shortest positive clause which appears in the most negative 2-clauses. Ties are broken randomly. It uses no value heuristic as it only tries the value TRUE for any variable. After each instantiation it performs unit propagation, by removing all positive clauses in which the chosen variable appears, and by removing from the positive clauses all the variables which appear with the chosen one in some negative 2-clauses. DCS uses a very carefully designed data structure which allows it to perform the unit propagation very fast.

[4]Any SAT instance can be converted to such a form by at most doubling the number of variables and the number of clauses.

The authors gave empirical evidence suggesting that DCS outperforms other state of the art complete SAT solver on SAT converted Dominating Clique Instances and direct encoded random SGI instances. They pointed out that DCS performs very badly on random k-SAT. Unfortunately there is no specification of other classes of SAT instances on which DCS would perform well.

Observation 4.8 The action of DCS can be viewed as performing Forward Checking on a BinCSP encoding of the original SAT instance. The encoding adds a variable for each clause. The values correspond to the variables of the original SAT instance. The binary constraints on any pair of CSP variables avoids the pairs of values which correspond to SAT variables occurring together in a negative 2-clause. The heuristic used by DCS corresponds to the Fail First Heuristic used by BinCSP solvers.

4.4 Comparison of random k-SAT

In this section we compare, under the same input length criterion, the typical hardness of random SAT having clauses of length 3, 4, 5 and 6. We use the running time of SAT solvers as a general measure of hardness, as it is reported by all solvers. When we compare the performances of the same solver we also use other parameters: number of visited nodes, number of steps, etc.

We use two complete solvers: satz(SZ) and kcnfs(KC) and two stochastic solvers esgi(ES) and novelty(NV). We did not consider random restarts for the stochastic solvers. The other parameters (rate of weight shrinkage for esgi; second best move and random walk for novel) were optimized for each random k-SAT using the methodology introduced in [9].

For each size and each value of k, we chose a range of values for clauses to variables ratio around the corresponding threshold. For each such triple of values

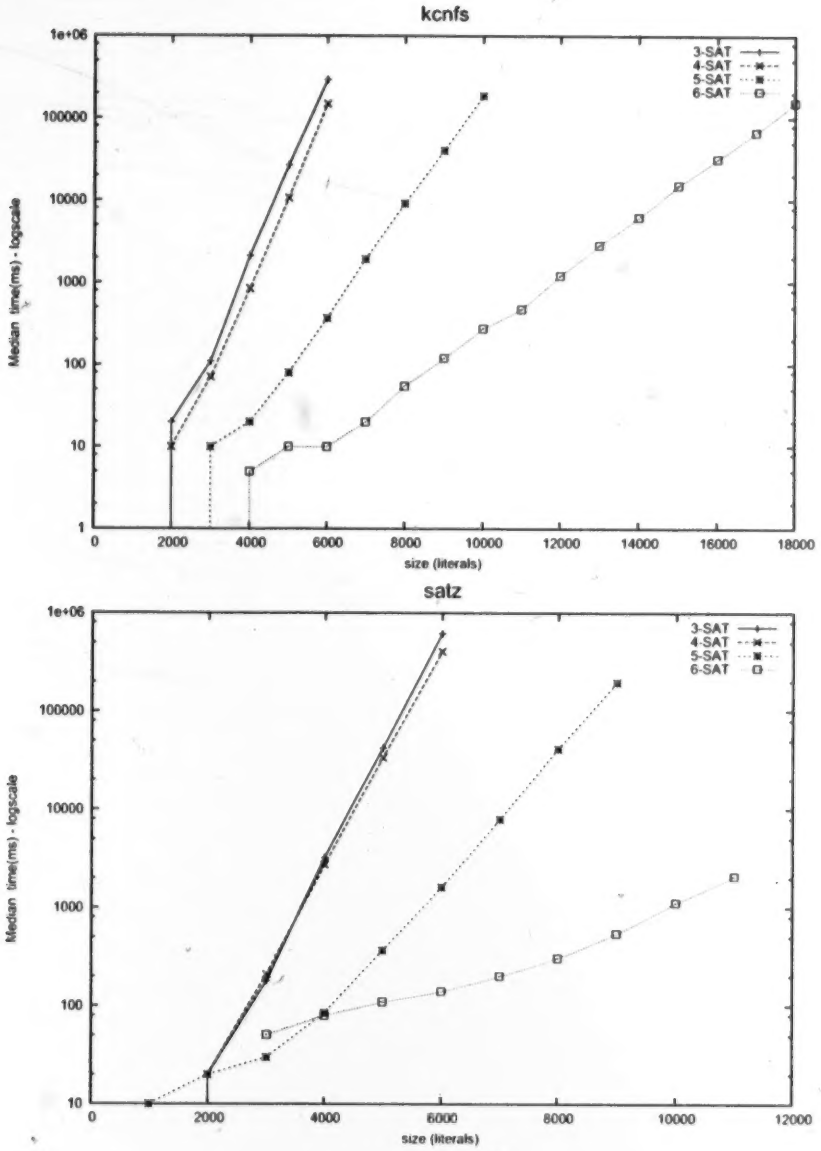

Figure 4.1: Evolution of hardness (expressed by time) with size for complete solvers

we generated 100 random instances. All solvers run on the same instances.[5]

For every k the running time of the complete solvers increases exponentially with the size of the instance. As presented in Figures 4.1 and 4.2 for all solvers the running time decreases with k. As the running time of stochastic solvers was pretty small we extended the experiments to larger sizes and the same conclusion was drawn: the running time decreases with k. We also compared other parameters reported by the solvers, such as number of visited nodes, number of branches, number of steps etc (see Figures 4.3 and 4.4). The plots suggest the same conclusion, even though, for small sizes, the number of nodes visited by kcnfs and satz seems to be smaller for random 3-SAT than it is for random 4-SAT. While it is absolutely clear that random 6-SAT is easier than random 5-SAT which in turn is easier than random 3-SAT and random 4-SAT, the hardness of random 3-SAT and random 4-SAT are close. We believe this is because the solvers are tuned to get the best results on random 3-SAT instances, and thus we conclude that random 3-SAT is the hardest of random k-SAT (at least for the size of instances SAT solvers can deal with).

Our experiment covers a small part of the size domain. Extending the range of size would require prohibitively large time limits. On the other side, asymptotic analysis requires exact formulas for the running time of the solvers. We escape this dilemma with a "complexity speculation" which indicates that under certain assumptions our conclusion may remain true on a larger range of sizes, even asymptotically.

In [2] it is proved that, with high probability, the running time of Ordered DLL on a random k-SAT instance having a clauses to variable ratio Δ slightly above the satisfiability threshold is $2^{\Theta(n/\Delta^{1/(k-2)})}n^{\pm\Theta(1)}$. Let α_k be a value slightly above the satisfiability threshold of random k-SAT and S be the size of the instance. The

[5]For stochastic solvers we filtered out the unsatisfiable instances. We had to stop satz early as it exceeded the time limit of 1200 seconds.

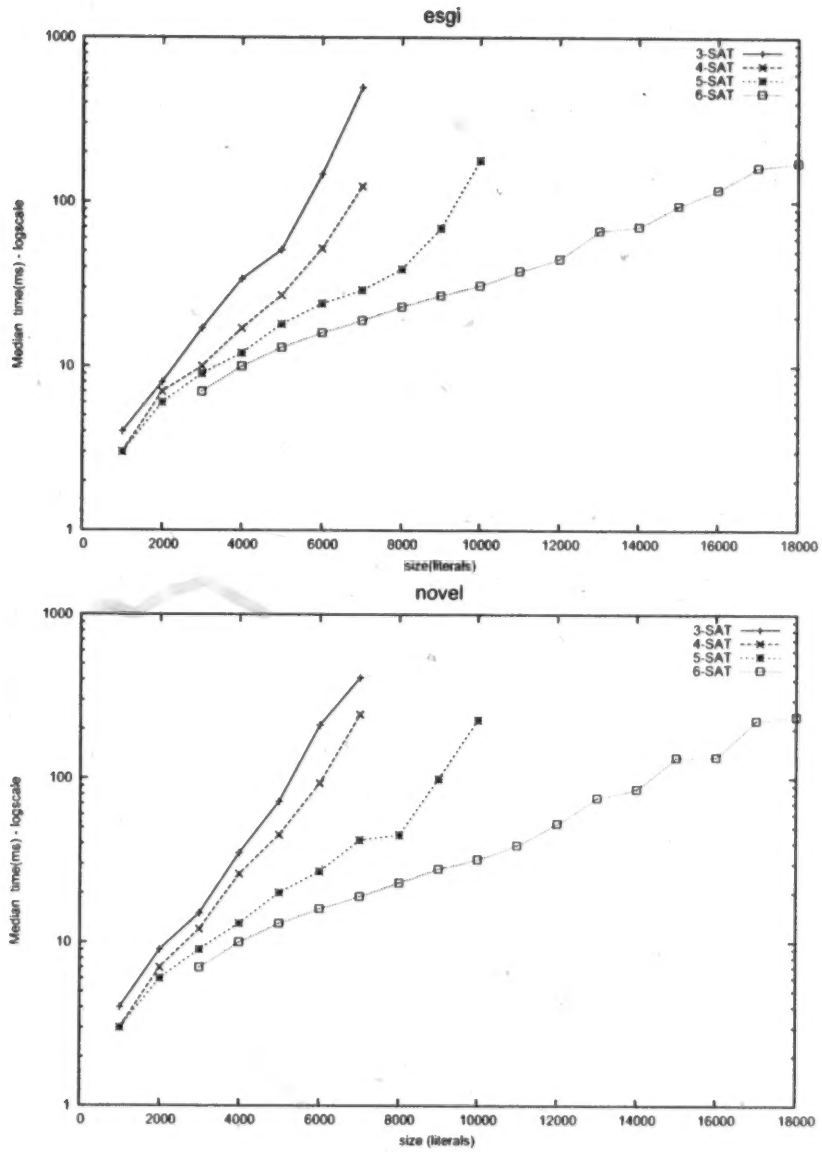

Figure 4.2: Evolution of hardness (expressed by time) with size for stochastic solvers

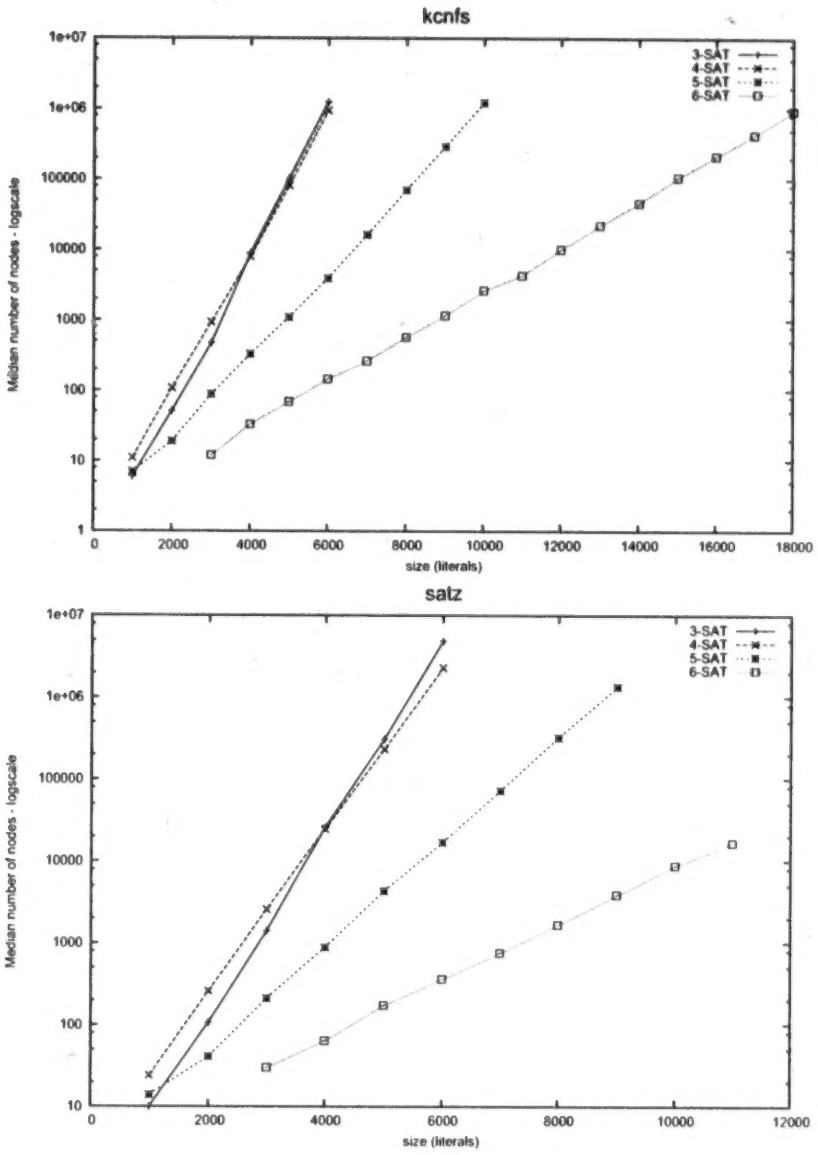

Figure 4.3: Evolution of hardness (expressed by number of nodes) with size for complete solvers

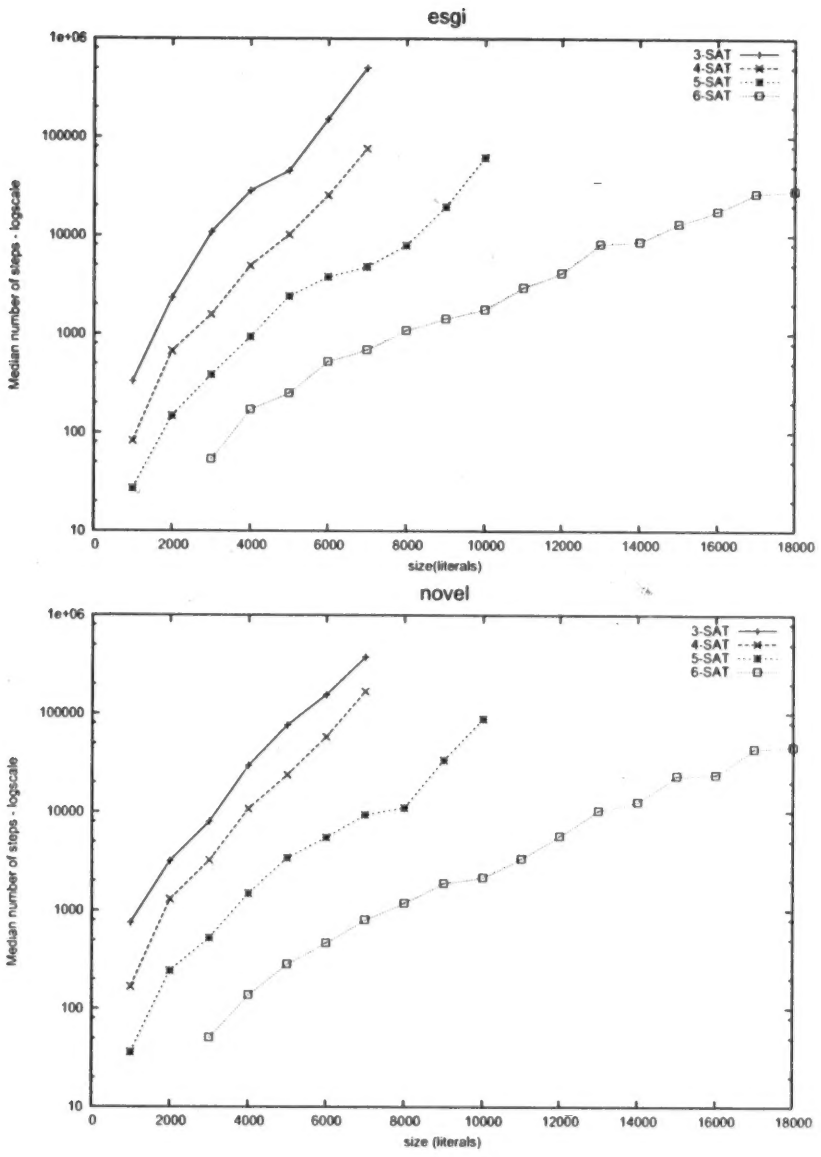

Figure 4.4: Evolution of hardness (expressed by number of steps) with size for stochastic solvers

running time of Ordered DLL on a random k-SAT can be expressed as:

$$T_{k,S} = 2^{\Theta\left(\frac{S}{k\alpha_k^{1+1/(k-2)}}\right)} \left(\frac{S}{k\alpha_k}\right)^{\pm\Theta(1)}.$$

Assuming comparable constant β in the first Θ approximation we obtain:

$$T_{k,S} = 2^{\frac{\beta S}{k(\alpha_k)^{1+1/(k-2)}}} \left(\frac{S}{k\alpha_k}\right)^{\pm\Theta(1)}.$$

Thus, for any $k \geq 3$ we have:

$$\frac{T_{k,S}}{T_{k+1,S}} = 2^{\beta S\left(\frac{1}{k(\alpha_k)^{1+1/(k-2)}} - \frac{1}{(k+1)(\alpha_{k+1})^{1+1/(k-1)}}\right)} \left(\frac{S}{k\alpha_k}\right)^{\pm\Theta(1)} \left(\frac{S}{(k+1)\alpha_{k+1}}\right)^{\pm\Theta(1)}$$

If follows that if

$$\frac{1}{k(\alpha_k)^{1+1/(k-2)}} - \frac{1}{(k+1)(\alpha_{k+1})^{1+1/(k-1)}} > 0 \tag{4.1}$$

then asymptotically (for $S \to \infty$) $T_{k,S} >> T_{k+1,S}$.

Equation 4.1 can be rewritten as:

$$1 + \frac{1}{k} - \frac{\alpha_k^{\frac{k-1}{k-2}}}{\alpha_{k+1}^{\frac{k}{k-1}}} > 0 \tag{4.2}$$

It easy to check that condition (4.2) holds for $k = 3, 4, 5$ and α_k chosen close to the threshold.

To complete the discussion we will use the following relation proved in [1]. For $k \geq 6$, the threshold value T_k of the random k-SAT satisfies:

$$2^k \ln 2 - 2(k+1)\ln 2 - 1 - 3/k \leq T_k < 2^k \ln 2$$

It follows that for $k \geq 6$, $2^{k-4}11 \ln 2 < T_k < 2^k \ln 2$. As α_k is chosen very close to the threshold we can assume that $2^{k-4}11 \ln 2 < \alpha_k \leq 2^k \ln 2$.

Therefore we have:

$$\frac{(\alpha_k)^{\frac{k-1}{k-2}}}{(\alpha_{k+1})^{\frac{k}{k-1}}} = \left(\frac{(\alpha_k)^{\frac{(k-1)^2}{k(k-2)}}}{\alpha_{k+1}}\right)^{\frac{k}{k-1}} = \left(\frac{\alpha_k}{\alpha_{k+1}}(\alpha_k)^{\frac{1}{k(k-2)}}\right)^{\frac{k}{k-1}}$$

$$< \left(\frac{2^k \ln 2}{2^{k-3}11 \ln 2}(2^k \ln 2)^{\frac{1}{k(k-2)}}\right)^{\frac{k}{k-1}}$$

$$< \left(\frac{8}{11}2^{\frac{1}{k-2}}(\ln 2)^{\frac{1}{k(k-2)}}\right)^{\frac{k}{k-1}} < \left(\frac{8}{11}\frac{10}{8}\right)^{\frac{k}{k-1}} < 1.$$

This shows that under the assumptions we made, Ordered DLL will find random 3-SAT harder than random 4-SAT and so on.

Observation 4.9 The situation is reversed for the Graph Coloring problem. 3-coloring of a random graph is the easiest of the random k-Coloring. The main difference is in the growing of the threshold with k. While for random k-SAT the threshold increases exponentially with k, for random k-Coloring it increases as $k\log(k)$.

4.5 Log-encoding vs direct-encoding for random SGI

It has been noticed that for several problems, CSP[8], Coloring[8], SGI, log encoding produces harder instances than direct-encoding. The empirical observation is based on comparing the encodings of the same "native instance". Not only is the log encoding harder but it also has less variables than the direct encoding. Thus under the same number of variables criterion the difference in hardness will be larger.

It is also known that typically log encoding generates larger, in number of literals, instances than direct encoding when applied to the same "native instance". Consider for example a SGI instance with a graph of order 32 and size 300 and a subgraph of

order 30 and size 250. By Observation 4.3, the direct encoding produces an instance having an input length of 254,560 literals. By Observation 4.5 the log encoding will produce an instance with an input length of 1,119,200 literals - more than four times larger than that of the direct encoding.

In this section we investigate the relation between hardness of log encoding and direct encoding of random SGI, using the same number of literals as comparison criterion.

We performed five series of experiments, one for each input length of 5,000, 6,000, 7,000, 8,000 and 9,000 literals. Each series has two sets of SGI instances, one for log encoding and one for direct encoding. Each set contains all random (n, m, p, q) SGI classes which, when converted to SAT by the corresponding encoding, result in instances having the desired input length.[6] For each class of SGI we generated 10 random instances. We ran satz, kcnfs, BerkMin(BM) and DCS(DC) on all instances of the direct encoding set, and satz, antisatz(AS), kcnfs and BerkMin on all instance of the log encoding set. *Antisatz* is satz with a reversed values heuristic - it always tries FALSE first. We used a time limit of 8 hours.

For each size, each encoding and each solver we selected the three classes of SGI which, when encoded to SAT, produced the hardest instances for the solver. We used the median of the running time to evaluate the hardness. When no more than 3 classes had 8 hours as median, we sorted them according to the number of instances solved. As expected kcnfs performed very poorly on direct encodings. Therefore for input length larger than 7,000 we could not determine the hardest classes for kcnfs as it timed out on almost all hard instances. For some classes of direct encoding instances of size 9,000, BerkMin stopped and reported "too many clauses", and therefore we report results on BerkMin up to 8,000 literals. In Appendix B we list

[6] As it is difficult to find SGI instances which correspond precisely to a SAT instance of a given size, we allowed a tolerance of $\pm 0.3\%$ for log-encoding and $\pm 0.015\%$ for direct encoding.

several tables which present for each solver the size and order of the graph and subgraph of the hardest classes as well as the median running time.

Figure 4.5 presents for each solver the evolution of hardness with size for the hardest, second hardest and third hardest classes.

Observation 4.10 We did not select the hardest instances of a certain size. We selected the classes which produced the hardest instances.

Figure 4.5 shows that under the same input length criteria, log encoding produces easier instances than direct encoding. The relation is very clear when the performances of the same solver are compared on log instances and direct instances. If the performances of the best solver for each encoding are compared than the conclusion is that direct encoding produces at least two orders of magnitude harder instances than log encoding. If we exclude satz on log encoding and kcnfs on direct encoding, then with very few exceptions the performances of the best solver for direct encoding, are worse than the performances of the worst solver for log encoding.

By comparing Figures 4.5 and 4.1 we noticed that, compared on the same input length, random 3-SAT is harder than direct encoding of random SGI.

In the absence of results regarding the location of the threshold for random SGI we have to rely on exhaustive search for finding the hardest instances. For this reason extending the experiment to larger sizes needs very long running times.

We noticed that satz performs very poorly on log encoded instances corresponding to SGI with subgraph of size one and graph of large size. Such SGI instances should be solved very easy, as the only requirement is to map the edge of the subgraph to an edge in the graph. We believe that the satz value heuristic is responsible for this abnormal behavior. Whenever it has to choose a value for a variable, satz tries TRUE first. We changed satz such that it always tries FALSE first. We called

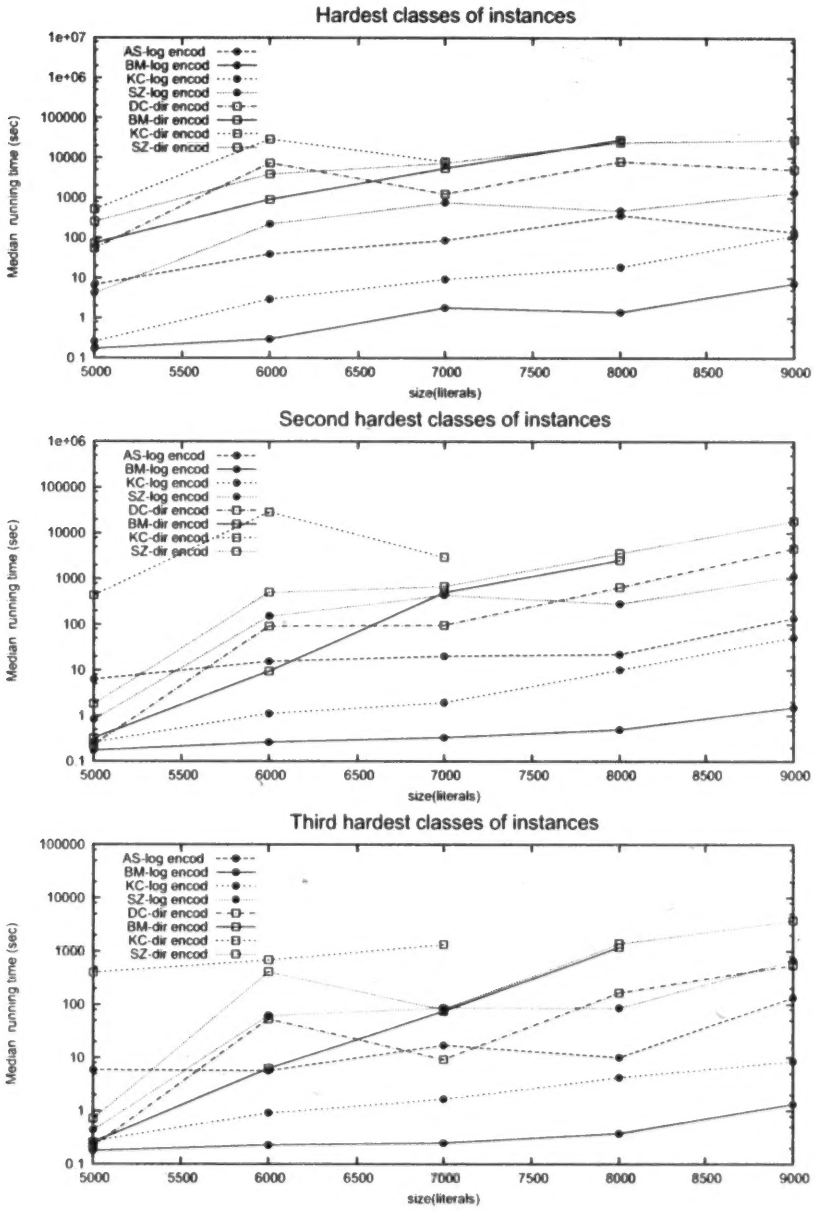

Figure 4.5: Evolution of hardness (expressed by time) with size for the hardest classes of log encoding and direct encoding of random SGI.

the new version antisatz. Antisatz consistently outperformed satz, in some cases (large instances) by one order of magnitude.

4.6 Log encoding of random SGI vs random k-SAT

It is generally believed that the log encoding produces hard SAT instances as it makes the SAT formula "flatter, featureless" [8]. The assumption is made under the same number of variables comparison criterion, and indicates that typically, log encoded SAT instances are harder than other SAT instances having the same number of variables, but still not as hard as the random k-SAT, for the corresponding k.[7]

In this section we investigate the relation between log encodings of random SGI and the corresponding random k-SAT, using the same input length criterion as base of comparison.

For $m = 16$, n from 8 to 15, and all possible values for p and q, we generated 20 random (n, m, p, q) SGI. For each n, we selected the two pairs of values for p and q which generated the hardest instances for Ullmann's algorithm. We ran satz, kcnfs and BerkMin on log encoding of the selected instances. The time limit was set to 2 hours. For $n \geq 12$ for several values of p and q, all solvers timed out without giving an answer. The size of the instances which could not be solved was between 25,000 and 50,000 literals. Therefore we only report the results for n between 8 and 11. The number of variables of the SAT instances varies from 32 to 44.

Table B.6 in Appendix B presents the number of variables and clauses of these instances and the running time of the solvers.

[7] As instances generated by log encoding tend to have clauses of size $2l_H$ they are usually compared with $(2l_H)$-SAT

Observation 4.11 While the chosen SGI instances are the hardest ones for Ullmann's Algorithm for SGI they are not necessarily the hardest ones for the SAT solvers. For example Ullmann's algorithm can quickly detect that the subgraph has a larger size than the graph, and report unsatisfiability. As the problem is similar to Pigeon Hole, a DPLL solver will need a long time to find a refutation of the log encoding of such an instance.

We ran the same solvers on instances of 8-SAT generated around the phase transition. The number of variables was chosen such that the input length of the instances varies in the same range as the log encoded instances. The complete results are presented in Table B.7 of Appendix B. [8]

Figure 4.6 presents solvers' running time on log encoded instances corresponding to the hardest and second hardest random SGI, and random 8-SAT. The conclusion is that log encoded instances are harder than 8-SAT ones having the same number of literals. When comparing the plots in Figure 4.6 it is important to know that the next point to be plotted for the log encodings has an x-axis value of about 25,000 literals and its running time is over 1200 seconds.

We noticed that satz was the quickest solver for log encodings and the slowest one for the 8-SAT. Therefore the comparison of satz running times on the two sets of problems becomes more meaningful, as it suggests that the time of the best solver on log encodings is two orders of magnitude higher than the time of the worst solver on 8-SAT, when instances of similar input length are compared.

While this is a singular example, it indicates that log encoded instances of random SGI are harder than the corresponding random k-SAT. The same conclusion may extend to other classes of problems which can be log encoded to SAT.

[8] BM would not solve 8-SAT instances with more than 34 variables and therefore we have no time to report for 35 and 45 variables.

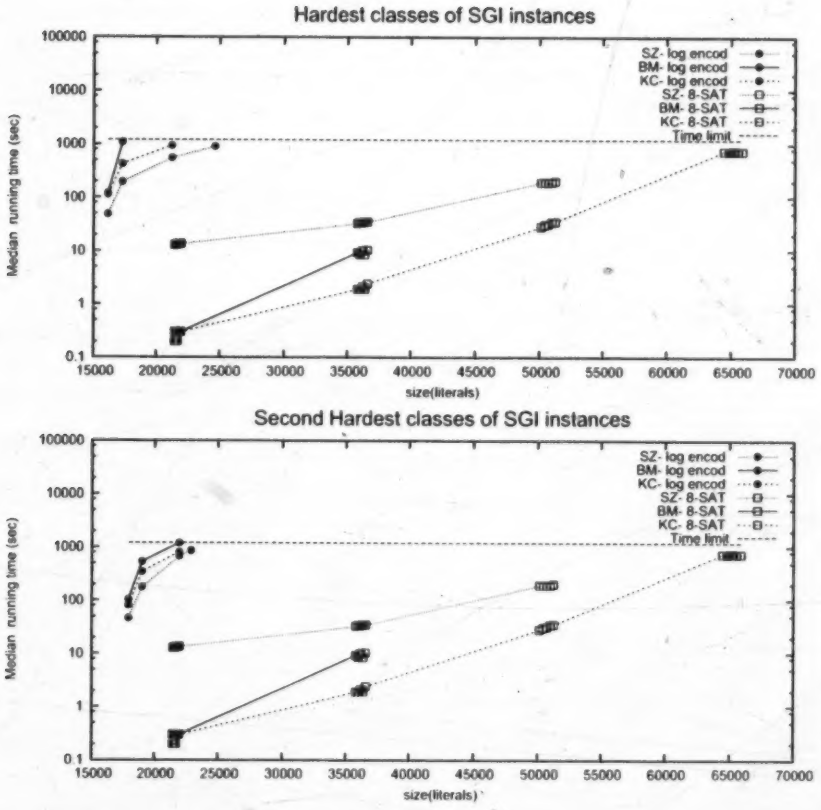

Figure 4.6: Evolution of hardness (expressed by time) with size for log encoding of hardest random SGI and 8-SAT.

Same number of variables
3-SAT $\preceq$ 4-SAT $\preceq$ 5-SAT $\preceq$ 6-SAT $\preceq \ldots$
3-SAT$\preceq$ direct encoded SGI $\preceq$ log encoded SGI $\preceq$ k-SAT for corresponding k
Same number of literals
$\ldots$ 6-SAT $\preceq$ 5-SAT $\preceq$ 4-SAT $\preceq$ 3-SAT
k-SAT for corresponding $k \preceq$ log encoded SGI $\preceq$ direct encoded SGI $\preceq$ 3-SAT
Same "native" SGI instance
direct encoded SGI $\preceq$ log encoded SGI

Table 4.1: "Contradictory" conclusions derived under different comparison criteria.

4.7 Conclusions

We considered and used the input size criterion for comparing CNF SAT instances. We motivated the use of this criterion with arguments related to empirical studies and complexity theory. Using this comparison criterion we obtained results which contrast common perceptions derived under different comparison criteria. The most important result is that random 3-SAT is the hardest of random k-SAT. For random SGI we presented empirical evidence that log encoding is harder than the corresponding random k-SAT but still not harder than direct encoding.

Table 4.1 summarizes the "contradictory" conclusions which can be derived under different comparison criteria. We used $A \preceq B$ to denote that instances from the hardest region(class) of A are empirically easier than instances from the hardest region(class) of B.

These "contradictions" suggest that the Artificial Intelligence community must start a systematic effort for defining sound, valid criteria for comparing instances hardness.

We recognize the limitations of our inquiry, as it is based, especially section 4.6,

on a small range of instance sizes. Despite these limitations, we consider that when different SAT models are compared, the same input length criterion should be used along with other standard criteria.

Bibliography

[1] D. Achlioptas and Y. Peres. The threshold for random k-SAT is $2^k \log 2 - O(k)$. *J. Amer. Math. Soc.*, 17(4):947–973, 2004.

[2] P. Beame, R. Karp, T. Pitassi, and M. Saks. The efficiency of resolution and Davis-Putnam procedures. *SIAM J. Comput.*, 31(4):1048–1075, 2002.

[3] D. Le Berre and L. Simon. Sat Competition Web Page. Available at http://www.satcompetition.org.

[4] D. Le Berre and L. Simon. The essentials of the SAT 2003 competition. In *Proc. SAT-03*, pages 452–467, 2003.

[5] J. Culberson, Y. Gao, and C. Anton. Phase transitions of dominating clique problem and their implications to heuristics in satisfiability search. In *Proc. IJCAI-05*, 2005.

[6] R. Dechter and F. Rossi. Constraint satisfaction. In *Encyclopedia of Cognitive Science*. 2000.

[7] I.P. Gent, H.H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proc. AAAI-99*, pages 654–660, 1999.

[8] H. H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. IJCAI-99*, pages 296–303, 1999.

[9] H. H. Hoos and T. Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112(1-2):213–232, 1999.

[10] D.G. Mitchell, B. Selman, and H.J. Levesques. Hard and easy distributions of sat problems. In *Proc. AAAI-92*, 1992.

[11] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):81–109, 1996.

[12] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. Assoc. Comput. Mach.*, 23(1):31–42, 1976.

[13] T. Walsh. Search in a small world. In *Proc. IJCAI-99*, pages 1172–1177, 1999.

[14] T. Walsh. SAT v CSP. In *Proc. of CP-00*, pages 441–456, 2000.

Chapter 5

Conclusions

We believe this research work has achieved its goal: to highlight the importance of finding and exploiting structural properties of Satisfiability instances.

We started by analyzing a subclass of generalized Satisfiability for which we established and proved a lower bound for unsatisfiability. The most important ingredients in obtaining the proof were the independence and uniform distribution of clauses. In other words it is the "absolute randomness" or complete lack of structure, which we exploited to derive the result.

Despite being random the *lk*-SAT model ensures a high local dependency between variables. The combination of these two characteristics allowed us to obtain rigorous upper and lower bounds for *lk*-SAT satisfiability threshold. We believe that the *lk*-SAT experiment suggests two important conclusions:

- there is little hope for designing an efficient "universal" Satisfiability solver. As *lk*-SAT shows, by slightly modifying *k*-SAT, we obtained instances which are very difficult for random *k*-SAT specialized solvers.

- it is very important to know and exploit any structural property of instances. In the *lk*-SAT case, adding P, the straightforward and fast preprocessing proce-

dure aimed at using the high local dependency between variables, significantly boosted the performances of the solvers.

The last subject we covered, comparing Satisfiability instances, proves the necessity of finding relevant criteria for comparison. Our inquiry showed that, by using the same input length criterion, well established beliefs are contradicted. We are aware that this criterion may draw similar criticism as the same number of variable criterion, but this only sustains our point: more work is needed to identify "good" comparison criteria.

It would have been nice if we introduced and analyzed such criteria in this dissertation, but any Ph.D. thesis must have an end, it here comes this thesis' end. But as the old saying goes: a Ph.D. thesis is not one's ultimate research work, it is rather one's first.

Appendix A

An approximation of the size of the instances for the trivial upper bound

In this appendix we try to estimate how big an instance must be for being trivially unsatisfiable. By equation (3.3) we have that for $m = (2^{k-1} - 1 + \varepsilon)n$, the expected number of conflicts is: $\varepsilon^2 n Pd(k,l)^2/2$, where $Pd(k,l) = \mathbf{Pr}[C \succ x]$ is the probability that the 2^{k-1} clauses generated on a cluster of size l imply a literal x. The estimated number of conflicts must be at least 1 for having the instance trivially unsatisfiable. This will result in a lower bound of $n_\varepsilon = \frac{2}{\varepsilon^2 Pd(k,l)^2}$ for n. As in the proof of Theorem 3.1 we will consider the independent version of the model.

We will approximate $Pd(k,l)$. Consider the case $k = l$. Let C be a cluster of size k on which 2^{k-1} clauses are generated. All clauses generated on C must contain the same literal x (which can be chosen in $2k$ ways), and all possible sign combinations of the remaining $k-1$ variables. As there are 2^{k-1} such combinations it follows that there are $2k(2^{k-1})!$ ways of choosing a sign combination which produces a unit clause. There are $(2^k)^{2^{k-1}}$ ways of choosing a sign combination. Thus, $Pd(k,k) = \frac{2k(2^{k-1})!}{(2^k)^{2^{k-1}}}$.

If $l > k$, than there are two phases in the process. The first one deals with selecting the 2^{k-1} subsets of l such that under the right signing they may imply a unit clause. The second phase deals with getting the right signing.

We will deal with the first phase first. The derivation of the unit clause from the initial clauses can be organized as a complete binary tree, where each node represents a resolution step, and each leaf represents an initial k-clause. Applying the required permutation to each clause, and then reordering the clauses, we can rearrange the clause set such that the frozen literal will be the first literal in all clauses; the literal on the second position will appear positive in the first 2^{k-2} clauses and negative in the last 2^{k-2} clause, and so on with all the literals.

There are $2l\prod_{g=1}^{k-1}(2(l-g))^{2^{g-1}}$ different trees over the set of l variables. As there are $k!(2^{k-1})!$ ways of getting a set of clauses from such a tree[1] there are at most

$$k!(2^{k-1})!2l\prod_{g=1}^{k-1}(2(l-g))^{2^{g-1}} = k!(2^{k-1})!2l2^{2^{k-1}-1}\prod_{g=1}^{k-1}(l-g)^{2^{g-1}}$$

sets of clauses that correspond to valid trees.

As we already considered permutations of clauses, for any given set of clauses which correspond to a valid resolution tree, there is only one sign combination that will freeze the variable.

Hence,

$$Pd(k,l) \leq \frac{1}{(2^k)^{2^{k-1}}}\frac{k!(2^{k-1})!2^{2^{k-1}}l\prod_{g=1}^{k-1}(l-g)^{2^{g-1}}}{2^{k-1}\binom{l}{k}^{2^{k-1}}} \leq \frac{k!(2^{k-1})!l\prod_{g=1}^{k-1}(l-g)^{2^{g-1}}}{(2^k)^{2^{k-1}}\binom{l}{k}^{2^{k-1}}}.$$

In table A.1 we present some values of n corresponding to a few combinations of ε, l, k.

[1] Not all permutations will result in distinct combinations of clauses, but as we need an upper bound we can over count.

k	l	$Pd(k,l)$	n_ε, $\varepsilon = 1$	n_ε, $\varepsilon = 0.5$	n_ε, $\varepsilon = 0.1$
3	3	0.035	1618	6,472	161,800
3	4	0.006	46,028	184,112	4,602,800
3	5	0.0006	4,994,360	1.9×10^7	4.9×10^8
4	4	7.5×10^{-5}	3.5×10^8	1.4×10^9	3.5×10^{10}

Table A.1: Size of instances for trivial unsatisfiability

Appendix B

Characterization of the hardest instances

B.1 Direct encoding VS log encoding results

In Tables B.1, B.2, B.3, B.4 and B.5 we present for each solver, each size, each encoding scheme and each of the three hardest classes the median running time and for the corresponding SGI instance:

- the order of the graph, n_H;
- the size of the graph, m_H;
- the order of the subgraph, n_G;
- the size of the subgraph, m_G.

It is worth noticing that each solver found as very hard some instances corresponding to SGI's in which the size of the subgraph is larger then the size of the graph. By the pigeon hole principle, such instances can be immediately refuted as unsoluble.

Hardness	Direct Encoding					Log Encoding				
	n_H	m_H	n_G	m_G	time	n_H	m_H	n_G	m_G	time
5000										
Hardest	13	73	12	63	73	12	6	7	3	0.2
2-nd Hard.	11	41	11	44	0.3	11	18	6	6	0.2
3-rd Hard.	13	51	8	27	0.2	16	88	6	6	0.2
6000										
Hardest	13	72	13	74	901	9	30	9	33	0.3
2-nd Hard.	14	89	14	89	10	9	27	9	22	0.3
3-rd Hard.	13	65	11	52	6	9	27	8	26	0.2
7000										
Hardest	14	86	14	85	5455	10	38	9	35	2
2-nd Hard.	14	83	13	71	500	10	34	8	26	0.4
3-rd Hard.	13	67	13	63	75	9	26	9	26	0.3
8000										
Hardest	15	99	14	88	27756	10	38	10	37	2
2-nd Hard.	15	96	13	76	2547	10	36	9	34	0.5
3-rd Hard.	14	83	14	85	1185	10	34	9	28	0.4
9000										
Hardest	NA	NA	NA	NA	NA	11	48	10	43	7
2-nd Hard.	NA	NA	NA	NA	NA	11	50	11	49	2
3-rd Hard.	NA	NA	NA	NA	NA	11	44	9	32	1

Table B.1: Characterization of the hardest instances for BerkMin

It is remarkable that for both encodings the best solvers were consistent in finding the same classes as the hardest ones. This suggests that the hardness is rather related to the SAT instance and not to some particular features of the solvers.

B.2 Log encoding VS k-SAT results

In Table B.6 for each of the hardest classes of random SGI instances used in section 4.6 we present the size of the subgraph, the number of variables of the corresponding log encoded SAT instances and solvers' the running times on these instances.

For the k-SAT instances used in section 4.6 we present, in Table B.7, the number of variables, the number of literals and the running time of the solvers. [1]

[1] BM would not solve 8-SAT instances with more than 34 variables and therefore we have no time to report for 35 and 45 variables.

Hardness	Direct Encoding					Log Encoding				
	n_H	m_H	n_G	m_G	time	n_H	m_H	n_G	m_G	time
5000										
Hardest	13	75	13	65	515	9	31	9	27	0.3
2-nd Hard.	13	73	12	63	439	9	18	6	13	0.3
3-rd Hard.	13	75	13	64	401	10	38	8	23	0.3
6000										
Hardest	14	89	14	88	>28800	9	30	9	33	3
2-nd Hard.	14	89	14	89	>28800	9	27	8	26	1
3-rd Hard.	13	72	13	74	683	10	32	7	20	0.9
7000										
Hardest	14	86	14	85	7881	10	38	9	35	9
2-nd Hard.	18	139	12	13	2986	10	34	8	26	2
3-rd Hard.	19	170	12	38	1334	10	39	10	33	2
8000										
Hardest	NA	NA	NA	NA	>28800	11	51	11	46	19
2-nd Hard.	NA	NA	NA	NA	>28800	10	38	10	37	10
3-rd Hard.	NA	NA	NA	NA	>28800	10	36	9	34	4
9000										
Hardest	NA	NA	NA	NA	>28800	11	50	11	49	117
2-nd Hard.	NA	NA	NA	NA	>28800	11	48	10	43	52
3-rd Hard.	NA	NA	NA	NA	>28800	11	44	9	32	9

Table B.2: Characterization of the hardest instances for kcnfs

Hardness	Direct Encoding					Log Encoding				
	n_H	m_H	n_G	m_G	time	n_H	m_H	n_G	m_G	time
5000										
Hardest	13	73	12	63	257	13	74	10	3	4
2-nd Hard.	13	75	13	65	2	13	74	9	18	1
3-rd Hard.	12	57	12	47	1	14	83	9	7	0.6
6000										
Hardest	14	89	14	89	3878	13	2	10	1	223
2-nd Hard.	14	89	14	88	507	13	4	10	1	154
3-rd Hard.	13	72	13	74	413	14	3	10	1	63
7000										
Hardest	14	86	14	85	7311	13	5	11	1	766
2-nd Hard.	14	83	13	71	689	13	7	11	1	444
3-rd Hard.	13	69	13	77	80	14	1	9	2	88
8000										
Hardest	15	99	14	88	24205	14	1	10	2	481
2-nd Hard.	15	96	13	76	3624	14	87	12	8	284
3-rd Hard.	14	83	14	85	1390	13	75	12	21	86
9000										
Hardest	15	99	15	103	> 28800	13	63	13	3	1375
2-nd Hard.	16	113	14	89	> 28800	13	15	12	2	1166
3-rd Hard.	15	96	14	87	3788	13	76	13	23	670

Table B.3: Characterization of the hardest instances for satz

Hardness	Direct Encoding				
	n_H	m_H	n_G	m_G	time
5000					
Hardest	13	73	12	63	55
2-nd Hard.	11	41	11	44	0.2
3-rd Hard.	12	57	12	47	0.2
6000					
Hardest	14	89	14	89	7345
2-nd Hard.	13	72	13	74	92
3-rd Hard.	14	89	14	88	53
7000					
Hardest	14	86	14	85	1244
2-nd Hard.	14	83	13	71	96
3-rd Hard.	13	69	13	77	9
8000					
Hardest	15	99	14	88	8111
2-nd Hard.	15	96	13	76	657
3-rd Hard.	14	83	14	85	165
9000					
Hardest	15	99	15	103	5132
2-nd Hard.	16	113	14	89	4609
3-rd Hard.	15	96	14	89	544

Table B.4: Characterization of the hardest instances for DCS

Hardness	Log Encoding				
	n_H	m_H	n_G	m_G	time
5000					
Hardest	11	53	10	26	7
2-nd Hard.	11	51	10	13	6
3-rd Hard.	11	42	10	4	6
6000					
Hardest	11	53	11	29	39
2-nd Hard.	11	26	11	2	16
3-rd Hard.	11	52	10	38	6
7000					
Hardest	14	1	9	2	86
2-nd Hard.	11	52	11	40	20
3-rd Hard.	11	47	11	15	17
8000					
Hardest	14	1	10	2	366
2-nd Hard.	11	51	11	46	22
3-rd Hard.	11	50	11	37	10
9000					
Hardest	13	76	13	23	141
2-nd Hard.	13	32	13	1	138
3-rd Hard.	13	77	13	45	135

Table B.5: Characterization of the hardest instances for antisatz

n_G	variables	literals	med. time SZ	med. time BM	med. time KC
8	32	16128	48.4	120.2	113.0
8	32	17920	46.3	102.3	80.6
9	36	17280	193.9	1082.2	425.7
9	36	19008	178.2	533.7	344.5
10	40	21248	551.1	$\geq$1200	932.7
10	40	21952	672.2	1183.5	801.4
11	44	24640	907.0	$\geq$1200	$\geq$ 1200
11	44	22880	858.0	$\geq$1200	$\geq$1200

Table B.6: Instance size and median running time (sec) for log encodings of the hardest random SGI - graph of order 16.

CVR	literals	% satisfiable	med time SZ	med time BM	med time KC
15 variables					
179	21480	63	12.8	0.2	0.3
180	21600	63	13.0	0.2	0.3
181	21720	46	13.1	0.2	0.3
182	21840	47	13.4	0.3	0.3
183	21960	52	13.5	0.3	0.3
25 variables					
179	35800	48	32.9	9.3	1.9
180	36000	47	33.9	8.6	1.9
181	36200	57	33.6	10.0	2.1
182	36400	47	34.4	8.4	1.9
183	36600	34	34.8	10.3	2.4
35 variables					
179	50120	41	193.4	-	28.3
180	50400	47	194.7	-	30.7
181	50680	50	192.5	-	31.56
182	50960	46	194.1	-	34.54
183	51240	30	201.3	-	35.32
45 variables					
179	64440	40	$\geq$ 1200	-	741.4
180	64800	45	$\geq$ 1200	-	733.1
181	65160	37	$\geq$ 1200	-	743.4
182	65520	28	$\geq$ 1200	-	738.5
183	65880	20	$\geq$ 1200	-	740.2

Table B.7: Instance size and median running time (sec) of 8-SAT at threshold.

B.3 Index of definitions and notations

Index

$ar(f)$, 4